Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr.rer.nat.)

Marco Lübbecke

ENGINE SCHEDULING BY COLUMN GENERATION

12. Juli 2001

1. Referent: Prof. Dr. Uwe T. Zimmermann
2. Referenten: Prof. Dr. Jacques Desrosiers, Prof. Dr. Sándor P. Fekete
eingereicht am: 16. Mai 2001

Für Kerstin, Gesa und Nele

Preface

There are many reasons,
rather than a single breakthrough.
—GEORGE L. NEMHAUSER (1994)

Mathematical optimization is a vital area in applied mathematics. During the few decades of its existence the field underwent not only theoretical advances. Today, we witness a change of focus towards making the large body of available methodology utilizable in practice. Much research efforts have been spent to tackle complex and large-scale real-world *applications*. A remarkable computational breakthrough was enabled by the development of sophisticated numerical and algorithmic techniques built on top of the theoretical fundament, often exploiting the structure of the respective problem at hand. It is not by chance that this evolution comes along with the enormous progress made in modern computer technology. The large amounts of data now available for the description of a practical problem at a realistic level of detail can reveal the limitations of an algorithm probably not detected in a laboratory environment. On the other hand, advanced implementations are the reason for the success of techniques known for a long time but used only recently. The ease of testing is a key feature offered by fast computers together with the ready availability of the most elaborated algorithms as commercial software. Thus, we are provided with an additional means of telling promising methods from futile ones.[1]

In view of this background *solving* a practical problem by means of applied mathematics is much more than applying mathematics. Implementation skills are of importance as well as the ability to communicate with practitioners and scientists from other disciplines. The proximity to and interaction with computer science and economics furnishes us with a spectrum of new ideas and techniques. This application oriented thesis deals with such a practical problem, *viz.* scheduling of locomotives, or *switching engines*, at industrial in-plant railroad companies. The problem is comprehensively introduced in Chapter 1. Column generation, our general solution paradigm, is a rich source of mathematical as well as implementation and application related

[1] We are aware of the fact that judging the quality of a theory primarily by its *impact* in practice embodies the danger of denying mathematics as a value by itself. However, it is the opinion of the author that recent trends will not replace the existing ones but rather complement and fertilize them.

issues. Chapter 2 is a self-contained overview. In Chapter 3 we discuss two model approaches; algorithms for solving the more promising of which are presented in Chapter 4. We conclude with computational results (Chapter 5), a more theoretical investigation of our problem (Chapter 6), and suggestions for an industrial implementation, and further research in Chapter 7.

We assume the reader to have a solid knowledge in linear programming, especially in duality theory. A basic familiarity with the fundamental ideas of solving integer programming problems will certainly prove advantageous. We also suppose some preliminaries to be known from graph theory and computational complexity theory. All of this (and more!) is found e.g., in the monographs by NEMHAUSER & WOLSEY (1988) and SCHRIJVER (1986). Notions and concepts inconsistently used in the literature are introduced when appropriate; *see* also the notation and symbol reference on page 175. We make use of different type faces to flag *newly defined words* (sans serif face) and words we personally feel deserve a certain *emphasis* (normal type italic) in the respective context. All claims and results developed in this thesis are proved. For results that are proved elsewhere the proof is omitted.

Acknowledgements

During the last three years, many people accompanied my work on this thesis in various respects. I would like to express my gratitude to my supervisor UWE ZIMMERMANN who introduced me to the field and community of discrete optimization, and who offered a fertile working environment, encouragement, and support ever since. I am obliged to JACQUES DESROSIERS and SÁNDOR FEKETE, who both instantly agreed to become co-referees for this thesis. A heartfelt thank you goes to MICHA BUSSIECK for always giving his ears and sympathy. I miss your shouting at me! My doctoral fellows HANNES SCHEEL and KATRIN VAN DER VEEN and our secretary HEIDI PFÖRTNER were there to get things off my chest. I very much enjoyed countless discussions with academic colleagues. I won't name you here; we all will meet for a beer again. Certainly, my family took on the hardest part—bearing my absence, and when I was present, my moods. Their love and patience made me stand the last months. *Ihr habt das alles erst möglich gemacht!*

My research was funded by the German Federal Ministry of Education and Research (BMBF) under grants no. 03-ZI7BR2-1, 03-ZIM2BS. All responsibility for the content is with the author.

Braunschweig, in May 2001 MARCO LÜBBECKE

Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Engine Scheduling

At night I wake up with the sheets soaking wet
and a freight train is running through the middle of my head.
—BRUCE SPRINGSTEEN, I'm on Fire

Many authors have attacked planning problems in *rail freight transport* by discrete optimization methods, *see* CORDEAU, TOTH & VIGO (1998) for a recent comprehensive survey. However, attention has been restricted to major railroad companies which operate nationwide, whereas we will deal with railroad companies located a industrial plants such as steel mills. To the best of our knowledge, the hierarchy of planning processes of such railroads is discussed here for the first time. In order to contrast our research with the literature, we sketch some basic problems in rail freight planning in the following section. In fact, important assumptions are quite different from those we present in Sections 1.2 through 1.4. In Sections 1.5 and 1.6 we elaborate the underlying formal structures.

1.1 Rail Freight Planning

The demand for rail freight transportation is usually expressed in terms of tonnage of certain commodities to be moved between origins and destinations, i.e., shippers and consignees. Traffic volume permitting, one will establish a direct connection between origin-destination pairs. Alternatively, and more commonly, cars pass through *classification yards*, where trains are split, and cars are reclassified, i.e., sorted and regrouped according to their respective destinations, incurring cost due to handling and delay. To prevent shipments from being reclassified at every

yard they pass through, several cars are grouped together to form a block. A block is attributed with its own origin-destination pair and cars are not regrouped on the corresponding leg. The *blocking problem* is to decide which blocks to build and which shipments to assign to which blocks. Recently, NEWTON, BARNHART & VANCE (1998) proposed a network design formulation to minimize total mileage, handling, and delay cost. When cars are assigned to blocks, blocks have to be assigned to trains. For instance, ASSAD (1980) discusses a non-linear multicommodity flow model for this so-called *makeup-policy*.

In order to avoid accumulations (and shortage) of empty cars at unloading destinations (loading origins), cars have to be repositioned in the rail network. The empty freight car distribution problem is strongly interconnected with the above, and various models have been proposed for its solution, most recently by HOLMBERG, JOBORN & LUNDGREN (1998).

Finally, tractive power has to be provided to perform the actual train movements. The operations research literature offers various synonyms for this planning stage, among these are *engine scheduling* (FLORIAN, BUSHELL, FERLAND, GUÉRIN & NASTANSKY 1976), *locomotive scheduling* (BOOLER 1980, 1995, FORBES, HOLT & WATTS 1991, WRIGHT 1989), *locomotive assignment* (ZIARATI, SOUMIS, DESROSIERS & SOLOMON 1999), and *scheduling of motive power* (WARDROP 1987). The problem is as follows. Given a set of timetabled trains, allocate one or more locomotives of compliant type(s) to trains at minimal operational cost. Usually, a periodic maintenance at appropriately equipped stations has to be respected as well. Most presented models have multicommodity network flow problem structure. CORDEAU, TOTH & VIGO (1998) give a more detailed problem formulation and comprehensively review the relevant literature.

In contrast to the above planning problems encountered at major domestic railroads, the literature on in-plant railroad operation has been scant. The first, and indeed the *only* (discrete optimization) paper, we are aware of, that deals with scheduling problems of *terminal switching railroads* is by CHARNES & MILLER (1956). A set of trips has to be covered by a set of routes such as to fulfill a certain demand for transportation. In fact, their model is of *covering type*. Despite its simplicity, the approach can be regarded as a preliminary stage of present complex vehicle routing models. What is more, the authors detail practical aspects which are definitively worth reading in our context. In the mid-fifties their judgment was that "complete and exact answers to the switching problem will probably have to wait until the current research by many investigators into DIOPHANTINE Analysis comes to fruition." Even today, in view of a *complete answer* this thesis is but a first step.

For us it is important to keep in mind that practically all of these planning problems refer to a *fixed schedule*, which is not given in our situation. We will see that this additional degree of freedom complicates matters considerably.

Remark. British and American English railroad technical terms differ partly significantly. We stick to American terminology whenever possible. The index contains a reference to the respective British counterparts, as far as known to the author. Much of the in-plant railroad related information discussed in this chapter was kindly provided by officers of the railroads involved in our project. We will use this information without further reference.

1.2 Industrial In-Plant Railroad Operation

Large industrial plants in the chemical, automobile, and steel industry often stretch over entire quarters of cities. As in the case of steel works heavy freight such as molten iron has to be transported between widely spread production, storage, or shipping terminals. In order to maintain a timely around-the-clock production process it may be indicated to operate a private railroad system as a distinct legal entity, often a subsidiary. As such, an *industrial in-plant railroad* is subject to competition[1] and has to be managed pursuant to economic aspects. Market deregulations in the railroad sector some years ago forced private railroads to offer a better transportation quality and to decrease charges. The efficient use—and desirably a reduction—of available resources became indispensable. Since the paper by CHARNES & MILLER (1956), apparently no attention in the operations research literature has been drawn on this kind of railroads. Reason enough, to shed some light on the involved combinatorial problems. The remainder of this section is entirely devoted to a better understanding of the every-day operation and the surrounding decision making processes of an industrial in-plant railroad.

Although the principles are generic, a steel mill serves as a good example for our exposition. Manufacturing of steel products such as chrome plated sheet is a multistage process. Melting iron, slab casting, cold reducing, or hot-dip galvanizing happens not only at different stages of the production process but also at different sites of the steel mill. From a railroad point of view these different sites or *terminals* are usually referred to as *customers* which are dichotomized according to their principal treatment of freight cars: *Loading terminals* request for empty freight cars suited for holding a specified intermediate, finished, or by-product, and *unloading terminals* order a certain quantity of raw materials or half-finished goods. The production plants of large industries may comprise several hundreds of such terminals, whose proper interaction has to be guaranteed by a customer oriented *freight car switching*. To this end, some industrial railroads run more than 100 *switching engines*, c.f. Figure 1.1 on the following page, to handle the flow of up to 6000 freight cars as depicted in the detail of a track layout in Figure 1.2 on page 5.

1.2.1 Hierarchy and Dependence of Planning Processes

Apart from being impracticable, at least these days, decision making in complex planning processes is not carried out in a monolithic fashion, neither by human planners nor by computer

[1] *Competition* in this context of course does not offer exchanging one railroad for another, but still, the management *is* exchangeable. And it is worth knowing that even the transport of molten iron *by truck* is possible today.

Figure 1.1: A switching engine of *Eisenbahn und Häfen GmbH*, Duisburg, Germany, transports molten iron at the steel works of Thyssen Krupp AG. These diesel or electric engines usually differ in their technical equipment, especially with respect to tractive effort, coupler, compressed air system, and radio control. Engineers have different skills and experience. Some need driving admissions for certain areas, particularly when driving on public tracks.

tools. In the simplest case, we have a *sequence* of decisions, often with feedback between single stages. Although from an optimization standpoint the hierarchical, or *decomposition,* approach wastes some optimization potential it usually reflects historically grown structures. For a fresh thinking on integrative perspectives *see* BORNDÖRFER & LÖBEL (1999). Figure 1.3 on page 6 gives an overview of the planning stages to be discussed next. As customary, we distinguish *strategic*, *tactical*, and *operational* issues according to the length of the respective planning horizon and the temporal impact and relevance of the decision. In brief, the three notions refer to planning in the long, mid, and short term, in that order.

The Customer's Planning

As Figure 1.3 suggests, the quantified customer's demand for transportation provides the raw data for all decision making in in-plant railroad operation. *Every* movement of freight cars refers,

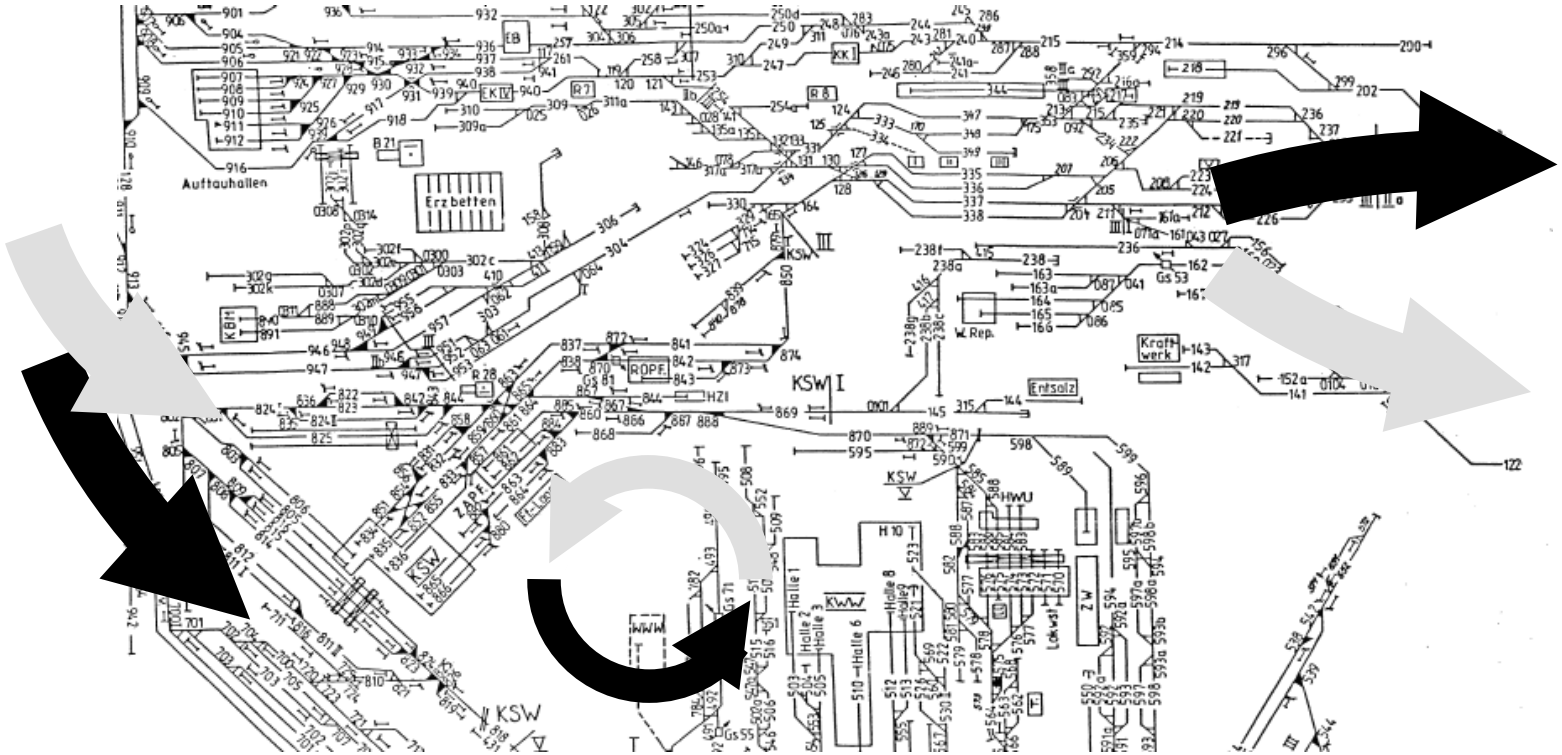

Figure 1.2: Inbound empty freight cars and raw materials, and outbound finished products and excess empty cars are interchanged with main line railroads like *Deutsche Bahn Cargo* in Germany, usually subject to a train schedule. Cars of inbound trains have to be decoupled and distributed according to their destination terminals. The moment when a car enters the in-plant track network is likely to differ considerably from the moment the customer is ready to process this car. The same is true for the reverse direction: Outbound rail cars from different customers but with a common shipping destination are intermediately kept on dedicated tracks before they leave the in-plant network at a later point in time. The problematic becomes especially obvious when cars are switched between loading and unloading terminals, or vice versa: It happens that freight cars which *have to* leave a loading terminal for some reasons are destined for a particular unloading terminal that is not yet ready to handle the cars. Analogously, yet unprocessed loaded cars at a certain unloading terminal may be needed elsewhere at a loading terminal. Although the overall production process of the steel works is coordinated, each customer is in a sense autarkic and has its own peculiarities.

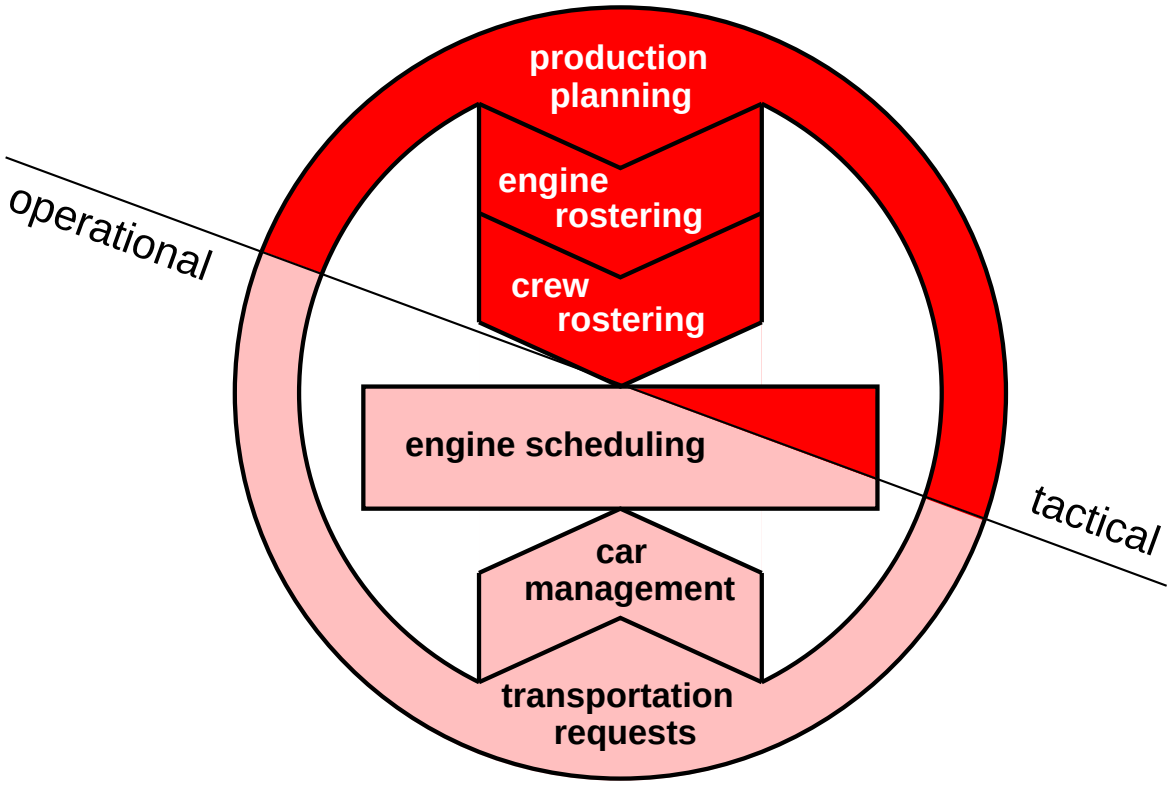

Figure 1.3: Hierarchy of operational and tactical planning processes. The customer's production planning, and request for transportation—represented by the circle—has direct impact on all actions of an industrial railroad—inside the circle.

directly or indirectly, to a particular customer. This is a significant contrast to public passenger transport systems being operated practically independently of *individual* demands. All the more important it is to strive for customer satisfaction by means of offering a *high transportation quality*, including e.g., timeliness. To set one spectacular example, an interruption of the blast furnace operation of a steel mill in consequence of an engine's delay may cause major financial damage, for which the railroad company may be liable.

Production Planning According to their supply contracts the customers devise a production schedule that determines the manufacturing program of each particular product. About a month ahead, this production scheduling is completed and serves as a mid term input for the railroad operation planning. This information is important inasmuch it constitutes an estimation of the expected quality and quantity of work, i.e., transportation, for particular working shifts.

Customer Requests Whenever an unloading terminal runs out of raw materials or when the capacity of cars at a loading terminal is almost exhausted this terminal *requests* for service, which

usually means exchanging the present cars for new ones. Apart from emergencies or unpredicted incidents no demand for transportation arises *suddenly*. However, there are customers who tend to issue their requests very late, even too late. Some of them request for *immediate* transportation although it is not operationally urgent. In such a case, the *dispatcher* is responsible for judging the priority of the respective request. As already mentioned, the customer information provided for each request usually specifies goods or cars in a certain quantity of a certain type only. Often only the *purpose* of the requested cars, say loading slag at the blast furnace, is transmitted and it remains up to the dispatcher to choose appropriate material, *viz.* cars and engines, to fulfil the request. Many tasks are obvious or regularly recurrent. They are taken into account by the dispatcher without explicit interaction with the customer, who expects this *automatic* generation of requests as part of the railroad's *service*. We describe the proceeding of the dispatcher in more detail below.

In-Plant Railroad Management Tasks

Strategic Planning Issues Long term decisions at industrial in-plant railroads involve purchasing or discarding engines and freight cars and re-designing the railroad track network. Although the network usually is historically grown it changes according to the changes and needs of the industrial plant it is located at. Nevertheless, for the rather short planning horizon we deal with in this thesis we may assume the infrastructure as well as the rolling stock and the operating personnel be fixed. Of course, deviations from the regular daily operation—like temporally closed tracks—may occur due to the *realization* of strategic measures and *have* to be considered in tactical and operational planning tasks. On the other hand, it is believed that there is an influence *upward* the planning hierarchy, *viz.* that an efficient planning at the tactical and operational level may result in cutting back the engaged resources and infrastructure in the long run, thus saving costs of the tied capital.

Engine Rostering The railroad's tactical planning process starts with the creation of a detailed engine roster for the next month. Based on the experience from previous months a standard roster is modified according to the peculiarities of the current planning period. The customer's precursory production schedule helps to forecasting the demand of engines of certain types for every working shift of the planning period. Note, that from now on the engines' places of employment are known. If necessary, engines are scheduled for regular maintenance at this stage.

Crew Rostering Once the engine rostering is completed engineers and accompanying personnel have to be assigned to each rostered engine. The size of the crew—in rare cases up to four members—depends on the type of the engine and on the work it is supposed to do. The personnel must be qualified and permitted to operate the respective engine. When it is necessary to travel along public tracks outside the industrial plant the engineer needs a valid permission for doing so. It is not only advantageous that the engineers be familiar with the area where the respective engine will be operated but also it may be mandatory to have certain qualifications or specific

knowledge when serving certain customers. Furthermore, collective agreements may regulate spare time claims, prescribed sequences of working shifts may have to be obeyed, and overtime hours are to be assigned fairly. Particularly in Germany, these regulations are hard constraints.

Freight Car Management Since the customers do not request for *particular* cars but for a certain transportation capacity or a specified input material the actual assignment of cars to requests is up to the dispatcher. This management of freight cars essentially hinges on the fact that only a small portion of the cars is actually owned by the railroad company. The majority of cars is rent from other railroads. Of course, cars are not identical and, especially in steel works, there are very special types of cars suited for a single purpose only. For all customer requests there usually is a certain type of car that fits best. However, it is often possible to replace certain types by others. From the customer's production schedule it is roughly known which amount of which type of car will be needed in particular working shifts. Although this information is already given at the tactical level, it is hardly possible for manual dispatching to anticipate the *exact* stock of cars at the operational level. In case neither the required quantity of the best fitting type nor one of the replacing types is available additional cars have to be rent. Since paying the cars is on an hourly basis the dispatcher tries to return the cars as soon as possible, which in turn influences his or her assignment of cars. This is only one of the conceivable scenarios.

Even having the required empty and loaded cars at hand does not imply that subsequent switching and transportation is uniquely predetermined. There may be several options to provide a customer with the requested cars, involving more or less switching and thus shorter or longer provision times. There is a tradeoff between expensive car rental and time consuming switching in order to make cars available. The goal is to minimize both the paid rent and the dead heading of cars while guaranteeing service without type mismatches. This problem is of a distinct combinatorial nature and deserves research investigations in its own right. However, this is beyond the scope of this thesis and will not be covered here in more detail.

1.3 Engine Scheduling and Transportation Requests

We assume the aforementioned planning tasks completed, i.e., engines and crews are rostered, and appropriate cars are available and dispatched. Since the final stage of *engine scheduling* by nature depends on the incoming requests, both issues are discussed simultaneously.

The customer's demand for transportation arrives sequentially at the dispatcher's working place by phone, fax, or radio transmission. These requests are gathered in a *pool*, ordered by non-decreasing due date which may range from a few minutes to several days ahead. Although this setting is inherently *dynamic*, it is legitimate to assume that for a planning period of, say, two hours our knowledge of future requests is (or can be made) sufficiently certain.

Hitherto, we were considering *customer requests* like the demand for exchanging loaded freight cars for empty ones at a loading terminal. One such request may induce several sub-tasks like switching and coupling in order to make up a train of empty cars, hauling this train to a load-

ing terminal, weighing the fetched loaded cars, and switching them to their next destination. An experienced dispatcher immediately identifies all necessary sub-tasks, in the sequel referred to as *transportation requests*. Actually, the notion of *transportation* is slightly misleading because we will subsume also tasks that do not involve the movement of cars. More precisely, we have four categories that cover the whole range of our understanding of transportation requests:

① **In-plant transportation** This type of request is the standard operation and refers literally to transportation from an origin track to a destination track. Since often coupling and decoupling, checking the compressed air system, or discharging of cars is required at both tracks, the total *duration* of the request is longer than just the period of transport.

② **Periodically scheduled trains** All traffic interchange with main line railroads is bound to a train schedule. Possibly necessary preparatory work has to be completed when a train is due. Obviously, all requests in this category are completely known even weeks ahead.

③ **Local operation** A terminal may require an engine to dwell for a certain while. For example during blast furnace tapping it is specified by safety regulations that an engine simply *stands by* in reach. Also, making up a train for a certain customer or a certain outbound direction may involve heavy, sometimes very time consuming, coupling, decoupling, and switching of cars. These are situations, when an engine stays at the same location, often at the same track.

④ **Exceptional and supplementary tasks** So far we assumed an engine to be *productive* all the time. Actually, this does not properly reflect reality. Engineers have to have a break within a prescribed time interval of their working shift. Engines may not be available due to scheduled or unexpected maintenance and repair. The same applies for fueling an engine. Shift changeovers are only possible at dedicated locations and generally prevent engines from serving customers at the same time. Apart from rare exceptions all requests in this category are known at least one working shift in advance.

The consensus of all these tasks is that the performing engine is *allocated* for an approximately known period of time; including preparatory work at the origin track and completing operations at the destination track. Such service may only be allowed during specified *time windows*, i.e., we have an earliest admissible commencement and a latest allowed completion of work. Such restrictions are determined e.g., by train schedules, berth periods of container ships, and deadlines for loading or unloading. Trains are weighed for the reason of cost accounting. Thus, the gross weight of the load is known, at least roughly.

Comparing a request's requirements with respect to technical equipment of the engine and skills of the personnel against the engines and engineers currently in duty immediately yields for each request the set of possible, or *admissible*, engines. At least *theoretically*, *all* engines may be admissible for particular requests. However, ordinarily, this set is constrained—for example non-availability of a catenary at a terminal requires a diesel engine, discharging of cars may require the appropriate air compression system, or fueling an engine restricts the set to precisely this engine. This list may be arbitrarily enlarged, but in either case checking admissibility is not

difficult to implement, for instance by means of a binary vector representation of required and available items, respectively, for each combination of requests and engines.

Based on the practical setting we will now start to elaborate a mathematical representation in order to work properly with the notion of transportation requests. We are mainly interested in a consistent treatment of all occurring cases. The attributes summarized in Table 1.1 suffice to capture all conceivable requests in the listed four categories. Given for a fixed planning horizon a set $\mathcal{R}$ of requests[2] and a set $\mathcal{E}$ of engines, let us reproduce the above information in an arc weighted directed graph[3] $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, c.f. Figure 1.4 on the next page. We will refer to $\mathcal{G}$ as the *request graph* for a given instance. The node set $\mathcal{N}$ comprises all tracks with relevance for engines or requests. More precisely,

$$\mathcal{N} = \bigcup_{r \in \mathcal{R}} \{r^+, r^-\} \cup \bigcup_{e \in \mathcal{E}} \{e^+, e^-\} \ ,$$

where e^-, $e \in \mathcal{E}$ are virtual tracks associated with the end of service of the respective engine. Note, that instead of *physical* locations which may be identical, nodes represent *logical* tracks which are all distinct. Two nodes $i \neq j \in \mathcal{N}$ are joined by an arc $ij \in \mathcal{A}$ if and only if sequentially visiting the logical tracks i and j, in that order, is operationally plausible for some engine. Briefly, service will not start with a destination track, will not end at an origin track, and the origin-destination sequence of a request must not be reversed. To be exact,

$$\mathcal{A} = \bigcup_{e \in \mathcal{E},\, r \in \mathcal{R}} \{e^+r^+, r^+r^-, r^-e^-, e^+e^-\} \ \cup \bigcup_{r_1 \neq r_2 \in \mathcal{R}} \{r_1^+r_2^+,\, r_1^+r_2^-,\, r_1^-r_2^+,\, r_1^-r_2^-\} \ . \qquad (1.1)$$

Although every sensible sequence of duties for an engine corresponds to a directed simple, i.e., node disjoint path in $\mathcal{G}$, the converse needs not be true. This issue will be dealt with rigorously in Section 1.5. Referring to the load to be picked up or delivered at a particular location r^+ or r^- for $r \in \mathcal{R}$ we will also use the notation $\ell_{r^+} = -\ell_{r^-} = \ell_r$. Note, that $\ell_{r^-} \leq 0$. Finally, let us shortly demonstrate the modeling flexibility of the attributes listed in Table 1.1. The request data, except for the tracks, are of course all optional and assume by default their respective largest degree of freedom, e.g., time windows are set to the entire planning horizon. If the particular location is of no importance, as for breaks, our logical tracks can be made *close to everywhere* by setting all respective t_{ij} to zero. An engine e may exclusively be reserved for stand-by or similar activity by setting $L_e = 0$. Also, time windows can be used to reflect high or low priorities. In fact, our mathematical understanding of a transportation request goes beyond what is customary used in practice.

Remark. The average velocity of an engine depends on whether cars are attached or not. Steep grades or small radii of bends may prevent a loaded, and only a loaded, engine from traversing

[2]Referring to transportation requests, from now on we will mostly use the short form *request*.

[3]All graphs in this thesis are *simple*, i.e., there exist no *loops* $(i,i) \in \mathcal{A}$, and no *parallel arcs*. Denoting arcs we will interchangeably use, both, $ij \in \mathcal{A}$ and $(i,j) \in \mathcal{A}$.

planning horizon	$\mathcal{R}$	set of transportation requests
	$\mathcal{E}$	set of available switching engines
network data, $ij \in \mathcal{A}$	t_{ij}	time required for traversing arc ij
	c_{ij}	cost incurred when traversing arc ij
engine data, $e \in \mathcal{E}$	L_e	tractive effort (i.e., capacity) of engine e
	e^+	(logical) track where engine e starts its service
	$[\underline{t}_{e^+}, \overline{t}_{e^+}]$	time window during which e becomes available
request data, $r \in \mathcal{R}$	$\mathcal{E}_r$	set of admissible engines for request r
	r^+, r^-	(logical) origin and destination tracks
	s_{r^+}, s_{r^-}	service time at origin and destination tracks
	$[\underline{t}_{r^+}, \overline{t}_{r^+}]$	start-of-service time window for origin track
	$[\underline{t}_{r^-}, \overline{t}_{r^-}]$	start-of-service time window for destination track
	ℓ_r	size of the load to be transported

Table 1.1: Summary of input data for an engine scheduling instance

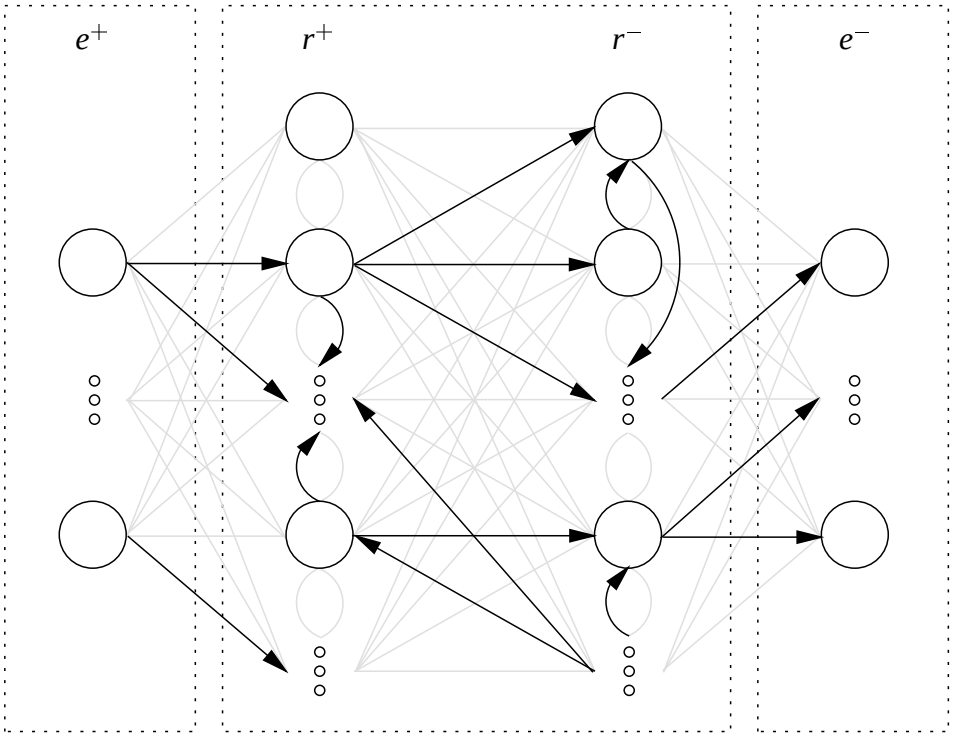

Figure 1.4: Schematic structure of the request graph $\mathcal{G}$

certain tracks. Other tracks may not be usable for certain types of engines due to technical constraints. However, we will not distinguish notationally between individual graphs and associated travel time and cost matrices, and assume that always the appropriate data, if available, is used. Of course, this is no problem in a computer implementation.

Current Manual Dispatching is Based on Passive Information Systems

To put it simple, engine scheduling is to allocate engines to transportation requests. Within noteworthy operational tolerances this fixes a sequence of requests with their associated service times for each engine in duty. In order to assure customer satisfaction, a punctual service must be enabled by the respective assignment for each engine. The tractive effort of an engine must never be exceeded. The scheduling process is decentralized such that each dispatcher is responsible for no more than ten or twelve engines, often significantly less. Still, a human judgment of dependencies between different decisions of such complexity is necessarily local and incomplete.

Although the dispatcher creates the schedules, their realization is beyond his or her control. The productivity of crews varies considerably as was already stated by CHARNES & MILLER (1956). The same operation will require different execution times for different crews, sometimes deviating by more than a factor of two. Moreover, weather conditions greatly influence the duration of transportation requests. In winter, cars have to stay in defrost facilities prior to being unloaded. Thus, large portions of the input data does depend on human experience and, at least these days, rest upon *estimations*. We will consider all human decisions regarding the input data as irrevocable.

Present computer aid is essentially limited to providing information on the status quo, e.g., a graphical display of the scaled track layout, the location of cars and engines, and relevant data about outstanding transportation requests, c.f. Figure 1.5 on the facing page. In a sense, a key functionality of these systems is to *track* the dispatcher's past decisions. We would like to refer to such kind of decision support as *passive*.

1.4 Computer Aided Scheduling

One might expect that in a workaday industrial operation an experienced dispatcher will *anticipate* many of the requests, and schedule the engines accordingly. While in normal operation this is actually true, under peak workload looking ahead is practically impossible. Then, requests are served on a first-come first-serve basis, or, more realistically, in a manner depictedly termed *loudest-shout first-serve*. For this reason, industrial in-plant railroads typically operate more engines than are actually needed in standard situations. For all railroads utilizing more than eight engines there is a considerable overhead of up to one fifth of all engines.

The question arose how to achieve a more regular and steady-going operation in order to mitigate these inefficient and resource consuming peaks. This motivated the incorporation of a *computer aided scheduling* tool into existing systems, which is able to offer an *active* decision

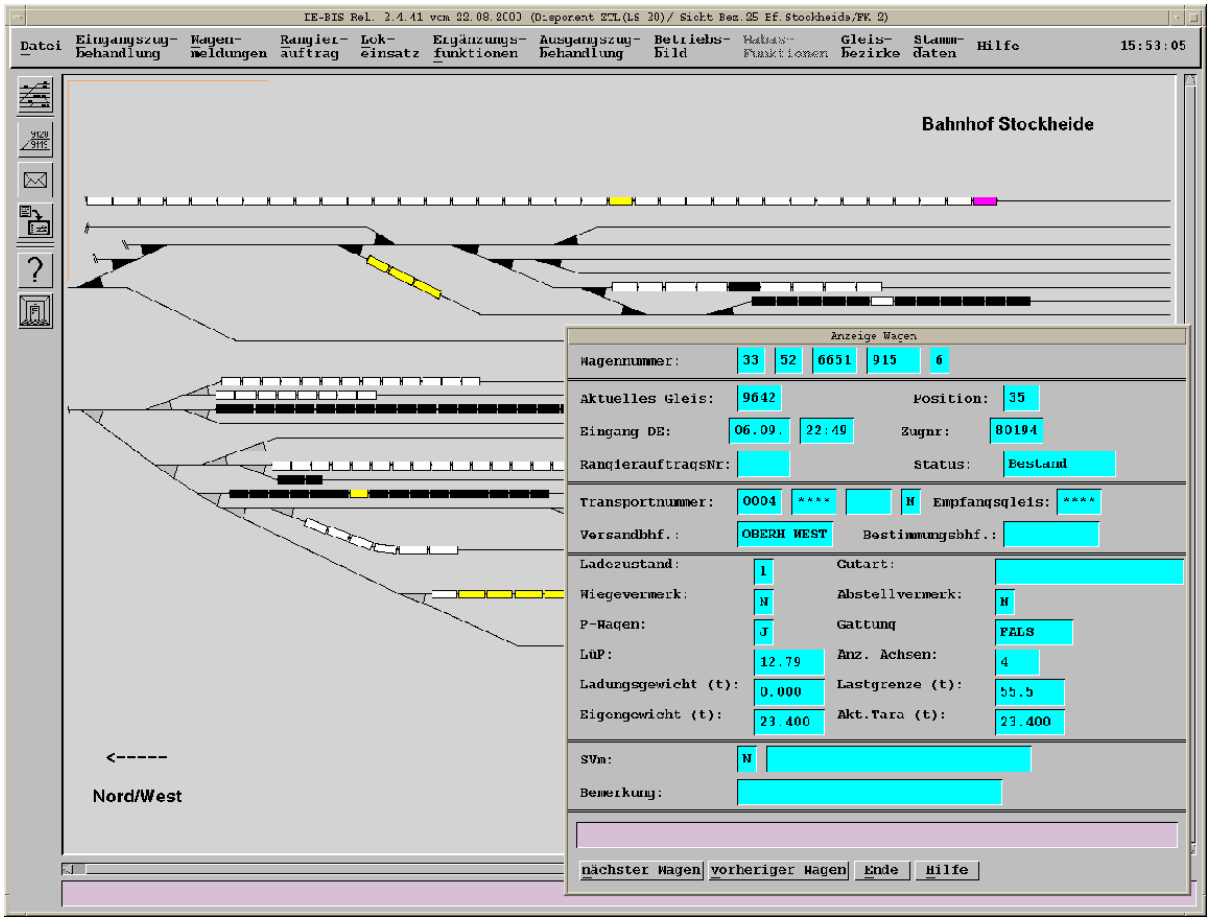

Figure 1.5: Screen shot of the information and dispatching software CP-BIS by CSC Ploenzke

support and relief for the dispatcher. Such a tool should submit scheduling suggestions that take into account *all* information available to the system. This enables one not only to individually respect the needs of particular customers but also to pursue operational, i.e., quantifiable goals. Note, that at present a schedule is of sufficient quality when it is feasible. A *global* view on the production process much better allows for exploiting the rationalization potential. BORND ÖRFER & LÖBEL (1999) detail further advantages compared to conventional planning.

Operational Goals

Although a reduction of the number of engines is aspired in the long term this will be achieved only mediately by improving the resource usage at the operational and tactical level. Conceivable measures include decreasing the total mileage traveled by the engines. In view of the unavoidable transport induced by transportation requests it could be better to increase productivity by reducing dead heading and waiting times instead.

Ideally, the respective gains should be exploited to the largest possible extent. In other words, with respect to a given operational goal, the desired computer aided scheduling tool should produce an *optimal scheduling suggestion*. It is one of the aims of our research to provide the necessary mathematical technology underlying such a tool, i.e., models, algorithms, and implementation considerations.

1.5 Pickup and Delivery Problems

Time constrained routing and scheduling problems received much attention in the operations research literature, *see* DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995) for a recent survey, and DESAULNIERS, DESROSIERS, IOACHIM, SOLOMON, SOUMIS & VILLENEUVE (1998) for a unified framework. Generically, when weights are to be transported in time from origins to destinations by a fleet of capacitated vehicles we speak of the *multiple-vehicle pickup and delivery problem with time windows* (*m*-PDPTW). For its relevance to our practical situation we will now access some characteristics in more detail.

Definition 1.1 (Pickup and Delivery Path)
Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ as defined above. Let $R = \{e^+, i_1, i_2, \ldots, i_\nu, e^-\} \subseteq \mathcal{N}$ an ordered set representing a directed simple path in $\mathcal{G}$ for an arbitrary but fixed $e \in \mathcal{E}$. R is called pickup and delivery path if and only if the following conditions hold.

(i) *Either $\{r^+, r^-\} \subseteq R$ or $\{r^+, r^-\} \cap R = \emptyset$, for all $r \in \mathcal{R}$*

(ii) *If $i_{\nu_1} = r^+$ and $i_{\nu_2} = r^-$ for an $r \in \mathcal{R}$ then $\nu_1 < \nu_2$*

(iii) *$\sum_{k=1}^{\nu_1} \ell_k \leq L_e$, for all $\nu_1 \leq \nu$*

(iv) *Let $T_1 = \max\{\underline{t}_{e^+} + t_{e^+ i_1}, \underline{t}_{i_1}\} + s_{i_1}$ and $T_{k+1} = \max\{T_k + t_{i_k i_{k+1}}, \underline{t}_{i_{k+1}}\} + s_{i_{k+1}}$, for $1 \leq k < \nu$. Then $T_k \leq \overline{t}_{i_k}$ for $1 \leq k \leq \nu$*

(v) *If $r^+ \in R$ then $e \in \mathcal{E}_r$*

The conditions (i) through (v) will be called *pairing*, *precedence*, *capacity*, *time window*, and *admissibility* constraints, in that order. Condition (iv) respects that arrival at a location is allowed before the corresponding time window opens, but this incurs a waiting time. A feasible solution to the *m*-PDPTW is a set $\{R_e\}_{e \in \mathcal{E}}$ of (node sets of) pickup and delivery paths such that

$$\dot{\bigcup}_{e \in \mathcal{E}} R_e = \mathcal{N} \ , \tag{1.2}$$

i.e., the paths *partition* the set of nodes. Actually, they induce a partition of the set of requests as well. Note, that $R_e = \{e^+, e^-\}$ conforms to Definition 1.1, which is required to account for the possibility of not using engine $e \in \mathcal{E}$.

Recent Research Topics

SOL (1994) and SAVELSBERGH & SOL (1995) comprehensively review several characteristics and the relevant approximative and exact solution approaches to the pickup and delivery problem in its variants. In particular, they develop the most general presentation of the *m*-PDPTW to date, which allows a multitude of pickup points and of delivery points, respectively, for each transportation request. DESAULNIERS, DESROSIERS, ERDMANN, SOLOMON & SOUMIS (2000) additionally provide a catalog of practical applications. Benefitting from these survey papers, we will only comment on some of the most recent literature. Details on earlier work will be deployed throughout the text whenever relevant to the context.

OERTEL (1997) considers the *m*-PDPTW with *transshipment* allowed at predefined locations at linear cost. To this end, transportation requests are split in two (at least), thus artificially enlarging the problem size. The additional problem arises that, at transshipment locations, vehicles can only pick up a load already delivered by another vehicle. This may even lead to deadlock situations. OERTEL describes criteria to prevent such situations and proposes adapted TSP heuristics to create solutions for instances of up to 100 customers.

The *dial-a-ride problem* (DARP) is an important special case of the *m*-PDPTW, in which transportation requests correspond to passengers. It arises in the transportation for the handicapped and the elderly. Thus, additional constraints are present e.g., the need for loading wheel chairs. An approximate solution to DARPs is indicated because of the large size of practical problem instances (up to 6000 requests). A common heuristic approach to vehicle routing problems is *cluster-first route-second*, in which the assignment of requests to vehicles, and the determination of the sequence of visited requests for each vehicle is not performed simultaneously but in two subsequent phases. Two *mini-clustering* approaches were recently suggested, both of which group small numbers of passengers into clusters according to heuristic criteria. This step is formulated as a set partitioning problem, c.f. Section 2.1.3. IOACHIM, DESROSIERS, DUMAS, SOLOMON & VILLENEUVE (1995) then solve an asymmetric *m*-TSPTW on the set of clusters, whereas BORNDÖRFER, GRÖTSCHEL, KLOSTERMEIER & KÜTTNER (1999) allow vehicles to wait before starting the next cluster. Again, the routing phase is modeled as a set partitioning problem in both papers.

RULAND & RODIN (1997) investigate the polyhedral structure of the polytope associated to the convex hull of incidence vectors of 1-PDP paths. They derive some classes of valid inequalities to be incorporated in a branch-and-cut algorithm. Their code is able to solve instances of up to 15 customers in reasonable time. From this the authors conclude that the strength of the identified valid inequalities is insufficient. To the best of our knowledge, this is the only study in polyhedral combinatorics which has been proposed so far; particularly the statement of facet defining valid inequalities is still outstanding.

As for many combinatorial optimizations problems, very few is known about the so-called *online situation* of pickup and delivery problems, in which the knowledge about future requests is incomplete. In practice, one is often badly advised with the acknowledged theoretical concept of competitive analysis for evaluating the performance of online algorithms. The lack of a satis-

factory theory, in particular for the PDPTW, will certainly be a research incentive for the years to come. ASCHEUER, KRUMKE & RAMBAU (2000) made first moves into this direction with their investigation of the online DARP. With respect to competitiveness they propose a best possible online strategy, called *smartstart*, to serve transportation requests without time windows in a metric space. Their objective is to minimize the *makespan*, i.e., the completion time of the last vehicle. HAUPTMEIER, KRUMKE & RAMBAU (1999) furthermore compare intuitive scheduling strategies under the assumption of an infinite planning horizon.

The classical nearest neighbor heuristic for the TSP relies on a measure of proximity of two requests. As some authors point out, such a measure is not that obvious in the PDPTW situation. IOACHIM, DESROSIERS, DUMAS, SOLOMON & VILLENEUVE (1995) introduce the concept of *neighboring requests* in the EUCLIDEAN plane. Two requests are said to be neighbors if their time windows overlap, the respective locations are "close" to each other, the travel directions do not exceed a specified angle, and savings in travel time can be realized by jointly serving both requests.

Last, but not least, attempts have been made to unify various vehicle routing problems, revealing the PDPTW as a special case of a general framework of considerable complexity. For instance, DESAULNIERS, DESROSIERS, IOACHIM, SOLOMON, SOUMIS & VILLENEUVE (1998) propose a non-linear mixed integer program which also encompasses the possible linkage between different vehicle routes, e.g., to represent precedence relations between requests. For computational experiments *see* IOACHIM, DESROSIERS, SOUMIS & BÉLANGER (1999).

Objective Functions

A multitude of objectives has been proposed, *see* SAVELSBERGH & SOL (1995) for an overview. Minimization of the *fleet size* is common for DARP situations in which vehicles are rent on a day-by-day basis. As mentioned above, dead heading and waiting times are undesired for our practical problem. Therefore, one may explicitly minimize these two quantities, or we may chose as objective the minimization of the total time needed for all the vehicles to execute their paths, also known as *route duration*. Both objectives possess certain practical disadvantages, c.f. Figure 1.6 on the next page. However, with a rolling planning horizon the former may prove more appropriate.

In many vehicle routing problems there exists a hierarchy of goals, e.g., primarily minimize the fleet size, and secondly minimize a mileage or travel time dependent goal. Practitioners from in-plant railroads were sceptical about primarily reducing the number of engaged engines. This might change when suggested schedules prove that on average a number of engines is not needed.

1.6 Concatenation: A Framework for Engine Scheduling

The *m*-PDPTW is a flexible model capable to cover such various applications as door-to-door transportation systems as well as managing the parcel collection and distribution of a parcel

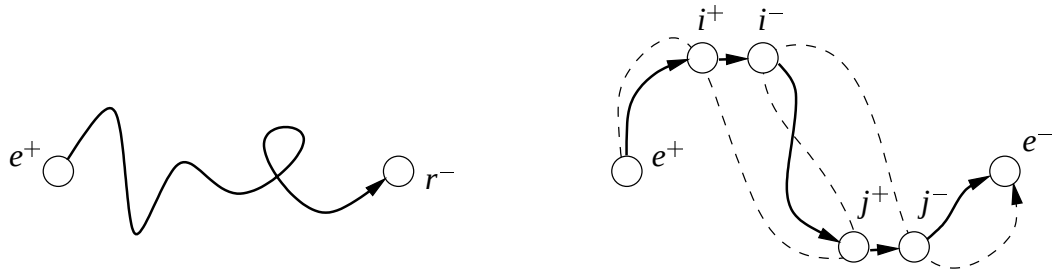

Figure 1.6: Pathology of objective functions. Consider minimizing the sum of route durations on the left. Request $r_{late} = \arg\max_{r \in \mathcal{R}} \underline{t}_{r^-}$ determines the route duration of the engine which serves it. Considerable dead heading may occur on this engine before r_{last} is served; schedules may be bad with respect to total travel distances. When dead heading time is explicitly minimized, c.f. the picture on the right, a related effect may occur. Instead of serving the depicted requests *i* and *j* consecutively, the two requests could be served as indicated by the dashed line. In effect, the engine runs less empty, but the mileage increases. At the worst, some request *r* is picked up, then serving other requests is for free (the engine is not empty all the time), and finally *r* is delivered.

service. While the generic model is definitively suitable for our practical situation, our additional knowledge about the problem structure is not adequately exploited. In a sense, the *m*-PDPTW is *too general*, in that it offers a flexibility we do not encounter in practice.

Railroads and Pickup and Delivery Problems

Railroad traffic imposes certain particularities on the *m*-PDPTW. At first, vehicle capacity is very restrictive. For instance, a switching engine is usually not powerful enough to haul more than two or three of the special type rail cars which hold molten iron, c.f. Figure 1.1 on page 4. Secondly, the loading and unloading operation itself is distinct. As a consequence of moving along railroad tracks the most reasonable loading scheme is *last-in first-out*. Clearly, deviation from this scheme incurs extra work in form of time consuming switching operations. Thus, in general the sequence of served requests will influence the individual processing times, which requires a warily choice of simultaneously served requests. This motivates the approach of restricting pickup and delivery paths to be built of a preselection of sensible loading/unloading sequences, leading us to the notion of precast request patterns.

Definition 1.2 (Pattern, Family of Patterns)
An ordered subset $P \subseteq \dot{\bigcup}_{r \in \mathcal{R}} \{r^+, r^-\}$ of nodes is called pattern if and only if there exists an $e \in \mathcal{E}$ such that the node sequence given by e^+, P, e^- is a pickup and delivery path. A family of patterns is a set $\mathcal{P}$ of patterns such that there exists a $\mathcal{P}' \subseteq \mathcal{P}$ with

$$\dot{\bigcup}_{P \in \mathcal{P}'} P = \dot{\bigcup}_{r \in \mathcal{R}} \{r^+, r^-\} \quad . \tag{1.3}$$

In order to access the set of requests visited by a pattern $P \in \mathcal{P}$ we use the notation

$$\text{requests}(P) := \{r \in \mathcal{R} \mid r^+, r^- \in P\} \quad .$$

The simplest non-empty pattern, which we will call *full truckload* for historical reasons, is $\{r^+, r^-\}$ for a given $r \in \mathcal{R}$, i.e., immediately delivering the load picked up in r^+. Note, that each node is allowed to be contained in different patterns of a family. Observe, that our definition of a pattern only predetermines the *precedence* or *sequence* of contained nodes, not the respective visiting times, i.e., the *schedule* itself. However, by Definition 1.1, for each pattern there exists a schedule which is feasible with respect to time window constraints. Moreover, in the modeling phase we will use the following.

Lemma 1.3 (Properties of Pattern Families)
A family $\mathcal{P}$ of patterns has the following properties.

(i) *For each $r \in \mathcal{R}$ there exists a $P \in \mathcal{P}$ with $\{r^+, r^-\} \subseteq P$*

(ii) *$\sum_{p \in P} \ell_p = 0$ for all $P \in \mathcal{P}$*

(iii) *If for any $r \in \mathcal{R}$ we have $r^+ = i_{p_1} \in \{i_1, \ldots, i_\nu\} = P \in \mathcal{P}$ then $r^- = i_{p_2} \in P$ and $p_1 < p_2$*

(iv) *$|\mathcal{P}|$ is finite but not necessarily polynomially bounded in $|\mathcal{R}|$*

Proof. Follows immediately from Definitions 1.1 and 1.2. □

Clearly, the family of largest possible cardinality is the collection of *all* pickup and delivery paths in $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. It is more interesting to restrict attention to a (small) subset of (simple) patterns, and use them to form more complicated paths. In the following we will do precisely this, exploiting the fact that a vehicle is empty after having visited all locations of a pattern.

Definition 1.4 ($\mathcal{P}$-Concatenation)
Let $\mathcal{P}$ be a family of patterns, and let $R = \{e^+, i_1, \ldots, i_\nu, e^-\}$ be a pickup and delivery path. R is called a $\mathcal{P}$-concatenation if and only if indices $1 = p_0 \leq \cdots \leq p_k = \nu$ exist such that

$$\{i_{p_0}, \ldots, i_{p_1}\}, \{i_{p_1+1}, \ldots, i_{p_2}\}, \ldots, \{i_{p_{k-1}+1}, \ldots, i_{p_k}\} \in \mathcal{P} \quad .$$

We will also refer to the act of constructing such paths as pattern concatenation.

Remark. Although this definition allows multiple visits to requests we will assume throughout this thesis that $\mathcal{P}$-concatenations be request disjoint unless otherwise stated. A concatenation with repetition of requests may illegitimately incur less waiting times or dead heading, and thus amplify the effects depicted in Figure 1.6. We will see later on that request disjointness has substantial implications on algorithmic design.

Figuratively speaking, by preselecting a family $\mathcal{P}$ of patterns we constrain each customer's *context* in admissible pickup and delivery paths. Note, that it is not known beforehand whether a given $P \in \mathcal{P}$ will be incorporated in any concatenation. Again, the *empty* concatenation $\{e^+, e^-\}$ for an $e \in \mathcal{E}$ conforms to this definition. We will particularly focus on concatenations made of the following families.

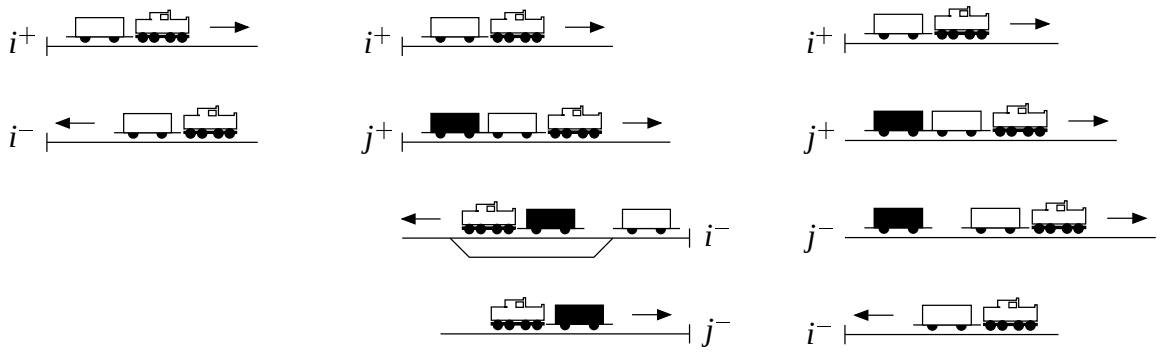

Figure 1.7: Typical situations for full truckload, overlapping, and embedding requests

Definition 1.5 (Regular Pattern Families)
The set $\mathcal{P}^k$ of patterns which visit exactly k requests out of $|\mathcal{R}|$, and which are not *concatenations of patterns $P \in \mathcal{P}^i$, $i < k$, is called k-regular pattern family, for $k = 1, \ldots, |\mathcal{R}|$.*

Note the technical particularity that (1.3) holds only if $|\mathcal{R}|$ is a multiple of k. Nevertheless, we will consider unions of such regular pattern families which will always admit feasible concatenations. From Lemma 1.3 (i) it is clear that each $P \in \mathcal{P}^k \neq \emptyset$ contains the origin and destination node, respectively, of exactly k requests. According to officers of in-plant railroads involved in our research, only the very simplest patterns do occur in practice, namely

$$\mathcal{P} \subseteq \dot{\bigcup}_{i \in \mathcal{R}} \{i^+, i^-\} \cup \dot{\bigcup}_{i \neq j \in \mathcal{R}} \big\{\{i^+, j^+, i^-, j^-\} \cup \{i^+, j^+, j^-, i^-\}\big\} = \mathcal{P}^1 \cup \mathcal{P}^2 \quad . \tag{1.4}$$

Reasons for this simplicity do not only emerge from the restriction to railroad tracks but also from the need for acceptable working conditions for the engineers. Figure 1.7 shows the structure of the three basic types of patterns which are called—besides the already known full truckload—*overlapping* and *embedding*, lending their names from the temporal relation of the involved requests. Actually, current manual dispatching constructs mostly full truckload patterns, and only a small fraction of patterns in $\mathcal{P}^2$. Note, that the choice (1.4) of patterns is not equivalent to allowing at most two requests to be served simultaneously on one engine, since for example $\{i^+, j^+, j^-, k^+, k^-, i^-\}$ for $i \neq j \neq k \in \mathcal{R}$ is not allowed. We will follow up some further matters related to $\mathcal{P}$-concatenations, which are not an issue here, in Chapter 6.

Given a family $\mathcal{P}$ of patterns, Definition 1.2 ensures the existence of patterns which partition $\mathcal{R}$. However, it is not guaranteed that there exists a partition which allows $\mathcal{P}$-concatenations to be constructed for all engines. For instance, time windows could be conflicting for patterns to be served on the same engine. Nonetheless, current (feasible) schedules suggest that the above choice of the pattern family (1.4) is reasonable and always allows a feasible solution. Capitalizing on the above definitions, and abbreviating from now on $\mathcal{P}^{1\cup 2} := \mathcal{P}^1 \cup \mathcal{P}^2$, we are now able to state our problem in a very compact form.

Definition 1.6 (Engine Scheduling Problem)
Given $\mathcal{E}$ and $\mathcal{R}$, a feasible solution to the engine scheduling problem (ESP) is a set $\{R_e\}_{e\in\mathcal{E}}$ of feasible $\mathcal{P}^{1\cup 2}$-concatenations such that their disjoint union visits all nodes, i.e., $\bigcup_{e\in\mathcal{E}} R_e = \mathcal{N}$.

Remark. The introduction of patterns allows for sequence dependent service and travel times (and cost). That is, we are able to account for the extent of work required at a particular location in dependence of the cars picked up previously. To simplify matters, we will not introduce a separate notation for individual arc weights. Again, this causes no difficulty in a computer implementation and we will assume that the appropriate data is used, if available.

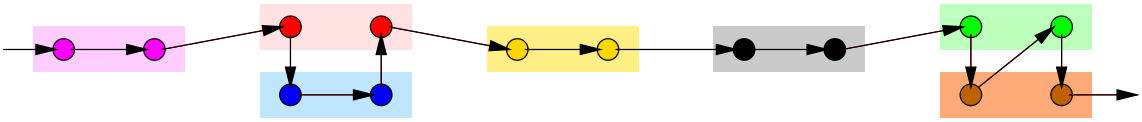

Figure 1.8: Structural appearance of a $\mathcal{P}^{1\cup 2}$-concatenation

Lemma 1.7 *For any $m \geq 1$, minimizing the sum of route durations for m-ESP is $\mathcal{NP}$-complete in the strong sense.*

Proof. Guessing a partition $\{R_e\}_{e\in\mathcal{E}}$ of $\mathcal{N}$ and checking whether each R_e, $e \in \mathcal{E}$ is a $\mathcal{P}^{1\cup 2}$-concatenation is polynomial in $|\mathcal{E}|$ and $|\mathcal{R}|$. Hence, *m*-ESP is in $\mathcal{NP}$.

Completeness in the strong sense follows from restriction to the *m*-TSP with time windows as follows. Split each city i of an *m*-TSPTW instance into $\{i^+, i^-\}$, and assign to both nodes the same time window as city i has, and service times of zero. Let $t_{i^+i^-} = c_{i^+i^-} = 0$. Arcs entering (leaving) city i in the *m*-TSPTW enter i^+ (leave i^-) in the *m*-ESP instance. The respective arc weights are preserved. Allow only $\mathcal{P}^1$-concatenations for the *m*-ESP, e.g., by defining $L_e = 1$, $e \in \mathcal{E}$, and $\ell_r = 1$, $r \in \mathcal{R}$. Clearly, an optimal solution to this instance immediately gives rise to an optimal solution to the corresponding *m*-TSPTW instance in polynomial time. □

The proof can be adapted to other objective functions as well. In fact, SAVELSBERGH (1985) proves that even finding a feasible solution to the TSPTW is $\mathcal{NP}$-complete in the strong sense. Therefore, by the same construction we have

Lemma 1.8 *For any $m \geq 1$, the problem of finding a feasible solution to m-ESP is $\mathcal{NP}$-complete in the strong sense.*

To the best of our knowledge, no research efforts have been undertaken in the direction of pickup and delivery paths of restricted combinatorial structure. This contrasts our experience that practically all computational experience reported on the PDPTW that we are aware of is drawn from problem instances allowing only a very special structure of paths. The interested reader

will verify for example in DUMAS (1985) and DUMAS, DESROSIERS & SOUMIS (1991) that the ratio

$$\rho = \frac{\max_{e \in \mathcal{E}} L_e}{\min_{r \in \mathcal{R}} \ell_r} \tag{1.5}$$

rarely exceeds very small integers, say $\rho \in \{2,3\}$. This observation is valid both for artificial and real world instances. Fairly large is $\rho \leq 5$ (SOL 1994), but also only $\rho = 1$ is considered (DESROSIERS, LAPORTE, SAUVÉ, SOUMIS & TAILLEFER 1988). Heuristically, patterns of "large" cardinality, i.e., patterns in $\mathcal{P}^k$, $k > \rho$, could be forbidden. Moreover, the practical situation might explicitly enforce precisely this—which is true for our application.

All cited authors conclude from their computational investigations that computation times decrease for highly constrained instances in terms of vehicle capacity. In dial-a-ride systems temporal constraints imposed by the customers strongly restrict the total vehicle load at any point in time, and the capacity constraints are of secondary importance (DESROCHERS, LENSTRA, SAVELSBERGH & SOUMIS 1991). Moreover, large fleet sizes are assumed by most authors. With as many vehicles available as half the number of requests (SOL 1994), we cannot obtain "complex" paths in any solution, even if, say, only one third of the fleet size is actually used. Also, clustering approaches produce on average mini-clusters of size less than four (DESROSIERS, DUMAS & SOUMIS 1987), or two or three, respectively (IOACHIM, DESROSIERS, DUMAS, SOLOMON & VILLENEUVE 1995). Thus, in consideration of practical needs, we feel a methodological exploitation of these facts is overdue. In fact, besides delivering a model and a solution approach to in-plant railroad engine scheduling, a first step is contributed by our research.

Distinguishing Properties of the Engine Scheduling Problem

In order to properly delimit the ESP definition we would like to describe some neighboring properties which do *not* hold. At first, the common notion of depot—i.e., a start and return point of each vehicle's itinerary—is void because open paths are usually sought instead of closed cycles. It is important to point out that the intention of $\mathcal{P}$-concatenations is *not* a cluster-first route-second approach in which *all* constructed clusters are actually built into routes. Instead, always a strict subset of $\mathcal{P}$ will be selected *simultaneously* to the construction of concatenations. Therefore, this approach is designed to be exact, provided $\mathcal{P}$ is inherently given by the problem.

Another contrast to the two stage strategy of cluster-first route-second is that patterns do not fix the temporal relation of nodes, i.e., travel times within a pattern. For instance, BORNDÖRFER, GRÖTSCHEL, KLOSTERMEIER & KÜTTNER (1999) force a vehicle not to wait within a cluster. Thus the total cluster duration is known in advance. This is reasonable in their DARP setting but would unduly constrain flexibility in our context. In particular, the ESP is not a fixed schedule problem altogether, i.e., node visiting times must be explicitly determined in a solution approach. This has known complicating consequences both on model building and algorithmic design.

Change in Problem Character

We deem it important to emphasize a consequence of our development in this section. Although originally based on the PDPTW, the ESP profoundly differs from the latter in its *problem character* due to the introduction of $\mathcal{P}$-concatenations. Pairing, precedence, and capacity constraints, respectively, which are essentials of the PDPTW are explicitly controlled by the precursory choice of a pattern family. Instead, besides time window and admissibility constraints which are common to both problems, the ESP is governed by decisions on selecting patterns that appropriately partition the node set $\mathcal{N}$. On the one hand, this strongly suggests to focus the model design on this selection. On the other hand, it inhibits, at least it aggravates the use of simple node oriented construction methods like straight forward adaptations of shortest path algorithms.

Further Applications of Pattern Concatenation

Empirical motivation for drawing attention to special structures of pickup and delivery paths comes from (1.5). We conclude with an outline of practical problems to which pattern concatenation is related, and where it may be a reasonable potential modeling alternative or complement. The references given are often only a representative for the respective problem class.

Ship Scheduling FAGERHOLT & CHRISTIANSEN (2000) consider seaborne transportation of various dry bulk cargos by a heterogenous fleet of ships in northern Europe. The problem structure is an *m*-PDPTW with a few loading ports and comparably more unloading ports. Each ship is equipped with a cargo hold which can be flexibly partitioned by means of variable bulkheads in order to lift different cargos simultaneously. The positioning of the bulkheads in one particular port may influence subsequent scheduling decisions, similar to the ESP where single schedule decisions have consequences for the amount of switching later on. Furthermore, the problem is well constrained in terms of compatibility between ship, port, and cargo particularities. Additional requirements concerning e.g., a safe trim of the load may be an issue. Also do coastlines impose restrictions on the sequence of ports to be visited. Consequently, the set of sensible loading/unloading patterns is severely constrained in its combinatorial complexity—a potential application area of pattern concatenation, possibly with an adapted definition of patterns, since ships need not necessarily become empty so often as our engines do. For the special case involving only one coastline we refer to Theorem 6.15 on page 149.

Extraction of Logs in Forestry RÖNNQVIST, WESTERLUND & CARLSSON (1998) report on an economically utmost important practical operative problem in Swedish highly mechanized forestry, *viz.* the extraction of roundwood from felling points to forestry roads in as short time as possible. Two types of vehicles are employed. *Harvesters* actually fell the trees and compile logs based on assortments which principally reflect the demand of the respective saw and pulp mills. These piles are collected by *forwarders* and moved to larger piles adjacent to forestry roads. Each forwarder starts its trip with empty driving to the first pile to pick up, and continues

loading logpiles until it is fully loaded. Then it returns to the appropriate pile (or piles) in order to get unloaded. Time windows are of no special importance. Several assortments can be loaded on a forwarder, however, only in predefined patterns. Figure 1.9 shows sectional drawings of different admissible such arrangements. Choosing one of the depicted patterns basically fixes the order in which piles are loaded and unloaded, e.g., because of risks the forwarder could tip over. It remains the decision which piles to actually load. However, for a given region this follows almost canonically from the way the harvest is done. With respect to our concept, pattern families could be devised for the sequence of loading each particular assortment in the respective regions, or even for each of the depicted arrangements of assortments.

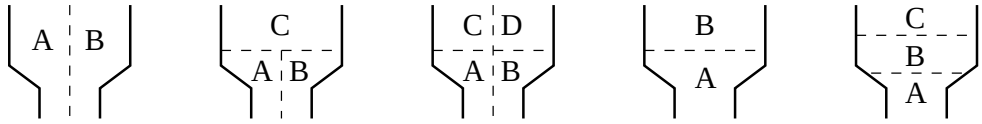

Figure 1.9: Arranging assortments of wood on a forwarder vehicle. The different letters indicate different assortments. Each assortment may be filled by loading more than one pile, when it is not too complicated, operationally. On average, five piles are loaded on one forwarding route.

Scheduling of Automated Guided Vehicles In order to meet the strongly increasing volume in maritime container transport, the *HHLA CTA GmbH*, Hamburg, Germany, plans to operate an almost fully automated container terminal. Concerning seaside operations, container vessels are loaded/unloaded via double rail mounted gantry cranes, and container are transported by means of uniform automated guided vehicles, or AGVs, to/from the intermediate stowage areas. AGVs navigate via electronic marks in the ground, thus, focus is on assigning and scheduling the transports. Interestingly, an AGV may lift either one forty feet container or two twenty feet container simultaneously. Therefore, each trip corresponds to serving either a 1-regular or a 2-regular pattern. Punctuality is a crucial factor to this operation, and what is more, permanently new information about e.g., container destinations is made available to the scheduler. This problem can be seen as an online and real time variant of the ESP.

Airline Schedule Generation ERDMANN, NOLTE, NOLTEMEIER & SCHRADER (1999) simultaneously determine aircraft rotations and passenger routes for an airline in the charter business. Since each aircraft basically alternates between a home country and some vacation targets, daily schedules, i.e., the respective sequences of direct flights, are often simply structured. This extremely simplified problem exposure is not originally in the pickup and delivery context, but an example for numerous network design applications. The general problem consists of determining a minimum cost flow in a network which is not *a priori* given but simultaneously constructed. Design decisions usually may involve nodes and arcs. In addition, we may involve a possibly broadened definition of patterns. It must be stated clearly, that building a pattern P is not equivalent to simply aggregating a subgraph $G' = (P,A)$ of the underlying graph G, where A denotes

the set of arcs induced by P in G. Aggregation would imply that if any arc in A is designed then this holds for all arcs in A, while in pattern concatenation arcs may appear in different patterns, and the implication is not valid. It is to be checked for the respective application whether a better exploited problem knowledge justifies the potential overhead involved by our approach.

CHAPTER 2

Selected Topics in Column Generation

It was just very very very big,
so big that it gave the impression of infinity far better than infinity itself.
—DOUGLAS ADAMS, The Hitchhiker's Guide to the Galaxy

Over four decades have passed since FORD & FULKERSON (1958) contemplated to deal only implicitly with the whole set of variables of a multicommodity flow model. DANTZIG & WOLFE (1960) pioneered this fundamental idea in a general setting, developing a strategy to columnwise extend a linear program as and when needed in the solution process. In their ground breaking papers on the cutting stock problem, GILMORE & GOMORY (1961, 1963) were the first to make actual use of the technique, soon to follow by e.g., APPELGREN (1969). Today, more than ever, *column generation* is a prominent—and sometimes the solely applicable—method to cope with linear programs[1] having a colossal number of variables.

Although the basic concept stayed the same, its environment continuously developed throughout the years. The embedding of column generation techniques within a linear programming based branch-and-bound framework paved the way for the exact solution of large-scale *integer* programs. Starting with the work of DESROSIERS, SOUMIS & DESROCHERS (1984) on routing problems with time windows, numerous practical applications were to follow. *Huge* programs to be solved only became *huger*, most drastically in the 1990s. Here, we will meet some challenges of tomorrow's *hugest* problems.

It is the aim of this chapter to provide a synopsis of the methodology according the author's personal preferences. It is partly close to, but not primarily intended for an inclusive literature

[1] Although the column generation context is much broader, we restrict ourselves to linear programming.

survey. In our presentation, there is a bias towards incorporating mainly recent material that has not found its way into textbooks, yet. Each of CHVÁTAL (1983), MINOUX (1986), NAZARETH (1987), and WOLSEY (1998), devote a chapter to the basics. LASDON (1970) is an amazingly valuable source of information on large-scale optimization. The surveys by BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998) and DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995), and the thesis of VANDERBECK (1994) are truly insightful. DESAULNIERS, DESROSIERS & SOLOMON (1999) is an indispensable computational reference.

Besides the above publications, there exists a column generation *jigsaw puzzle* scattered all over the vast literature. More than one half of this thesis' bibliography entries are referred to in this chapter—compiling 117 citations in total. As of this writing, some new fascinating ideas are available as technical reports only. A good deal of the findings related to our "selected topics" are nowhere stated as provable theorems, but hidden between the lines as observations and remarks. Whenever possible we try to merge kindred contributions, preferably making the outcome an explanation for empirical phenomena. Particular applications or even single problem instances gave the impetus for some interesting developments. Sometimes authors refer to *intuition*, and indeed, so do we occasionally.

It is not easy, if not impossible, to discuss the components of a column generation scheme separately or independently. More explicitly, our "selected topics" are strongly interconnected. An introductory methodology outline will help us to get to grips with the dependencies. Nevertheless, some concepts and notions will be casually used before their proper introduction. In this chapter, we are mainly concerned with the *linear* programming setting. However, throughout the whole chapter it is instructive to keep in mind that our ultimate aim usually is to solve an *integer* program, although the presentation of the procedure for doing so, branch-and-price, is deferred to the very end. As linear programming techniques are indispensably located at the heart of integer programming solution approaches, the *classical* methodology of *linear programming column generation* is the most important building block of integer programming column generation.

Arguably, there are no new results in this chapter. Still, we are not aware of a comparable accumulation of various sources, enabling us to find better ways to *relate* things. Hopefully, the chapter will successfully do this, conveying a clear understanding of fundamental and some more advanced ideas in column generation. They are the key to the success of the approach developed in this thesis.

2.1 Decomposition and Extensive Reformulation

Integer programs and their associated linear relaxations encountered in applications almost always exhibit a large deal of *structure*. Constraint matrices are typically very *sparse*, having non zero elements in the order of magnitude of one percent, or less. This well known phenomenon is due to the fact that activities associated with variables are subject to only a few of the conditions represented by the constraints, respectively. On top of that, there may exist a hierarchical, geographical or logical segmentation of the underlying problem, which is reflected in the model formulation. Thus, it is likely that the non zeros are grouped in such a way that *independent sub-*

systems of variables and constraints result, possibly *linked* by a distinct set of constraints and/or variables. Figure 2.1 shows schematically the distribution of non zeros for different angular *block diagonal* matrices, occurring in practice most frequently (MINOUX 1986).

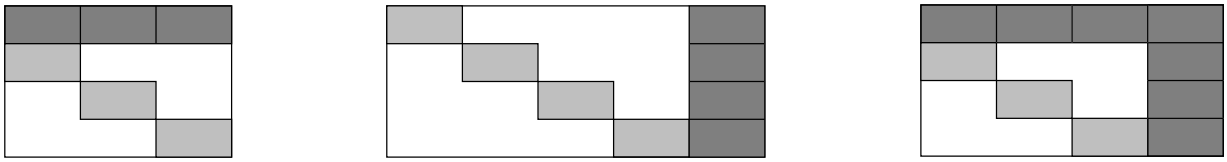

Figure 2.1: Block diagonal matrix structure, (a) with linking constraints, (b) with linking variables, and (c) with linking constraints and variables. Only the shaded regions may contain non-zero elements.

The general idea behind the *decomposition paradigm* is to treat the linking structure at a superior, *coordinating*, algorithmic level and to independently address the subsystem(s) at an subordinated level, exploiting its structure. Here, we are concerned with linking constraints only. By the way the aforementioned coordination is handled, the resulting approaches are usually called *price directive*. *See* e.g., the techniques of BENDERS (1962) and VAN ROY (1983), respectively, for the treatment of the other two cases. Consider the following general problem

$$\begin{array}{rrcl} \min & c(\boldsymbol{x}) & & \\ \text{subject to} & A\boldsymbol{x} & \leq & \boldsymbol{b} \\ & \boldsymbol{x} & \in & S\ , \end{array} \qquad \text{(CF)}$$

where $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^m$, and $S \subseteq \mathbb{R}^n_+$. The cost function $c : \mathbb{R}^n \to \mathbb{R}$ is not required to be linear. For the purpose of this chapter we will refer to (CF) as the original or *compact formulation*. The seeming distinction of different kinds of constraints allows us to uniformly treat different variants of this problem. It should be mentioned that such a *decomposition* can be applied to *any* (linear) program, since constraints and variables can always be partitioned, albeit artificially, into subsets. We focus in this section on strategies that exploit structural properties of the constraint matrices and allow for a *reformulation* of our compact model which is easier to solve or which offers other advantages. Later on we will show how to actually solve the resulting reformulations by column generation, a methodology inseparably connected to the decomposition approach. We will especially take into consideration the benefit of the discussed techniques for the solution of large *integer* programs.

Before we start, let us make some general remarks. Models with a large number of variables, to which we refer in this context as *extensive formulations*, do quite naturally arise in practical problem settings. There are occasions, where the models get that big that they cannot be stored in core memory[2] even of today's supercomputers. Cutting stock problems, which drew the research interest to column generation techniques, are only one example, *see* VANCE (1998) for a recent reference. Besides these *unavoidably* large models, there are situations when there is choice

[2]There may be other bottlenecks necessitating column generation techniques. SANKARAN (1995) reports on a specific application where the LP solver, `LINDO`, is restricted to handling only a limited number of variables.

between a compact and an extensive formulation. Since the former clearly outperforms the latter in terms of the size of representation the motivation for not preferring the compact one is not obvious at all. BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998) give reasons for considering (mixed) integer programs with a huge number of variables.

- Column generation provides a decomposition into master and subproblems known from the ideas of DANTZIG & WOLFE (1960). This decomposition may have a very natural interpretation in terms of the compact original problem, thus allowing for the incorporation of additional, possibly complicated, constraints.
- The extensive formulation may be stronger than the compact formulation in the sense, that its linear programming relaxation gives a tighter approximation to the convex hull of integer points. In the following we will provide a theoretical justification for this assertion.
- A compact (mixed) integer program may exhibit a symmetric structure which drains the performance of branch-and-bound algorithms, c.f. Section 3.1.1. The decomposition approaches discussed here suitably reduce or eliminate these difficulties, incurring bigger efforts necessary to solve the respective linear programming relaxations.

2.1.1 The Decomposition Principle in Linear Programming

To begin with, we briefly refresh the classical *decomposition principle* in linear programming, introduced by DANTZIG & WOLFE (1960), and today being part of the mathematical programming standard repertoire. As in the original paper, we assume $c : \mathbb{R}^n \to \mathbb{R}$, $c(\boldsymbol{x}) = \boldsymbol{c}^\mathsf{T}\boldsymbol{x}$ with $\boldsymbol{c} \in \mathbb{Q}^n$ and $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid D\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{x} \geq \boldsymbol{0}\}$. The well known theorems of MINKOWSKI and WEYL, *see* e.g., SCHRIJVER (1986), enable us to represent each $\boldsymbol{x} \in S$ as convex combination of extreme points $\{\boldsymbol{p}_q\}_{q\in Q}$ plus non-negative combination of extreme rays $\{\boldsymbol{p}_e\}_{e\in E}$ of S, i.e.,

$$\boldsymbol{x} = \sum_{q\in Q} \boldsymbol{p}_q\lambda_q + \sum_{e\in E} \boldsymbol{p}_e\lambda_e, \quad \sum_{q\in Q} \lambda_q = 1, \quad \boldsymbol{\lambda} \in \mathbb{R}_+^{|Q|+|E|} \tag{2.1}$$

where the very same theorems tell us that the index sets Q and E are finite. Substituting for $\boldsymbol{x}$ in (CF) and applying the linear transformations $c_j = \boldsymbol{c}^\mathsf{T}\boldsymbol{p}_j$ and $\boldsymbol{a}_j = A\boldsymbol{p}_j$, $j \in Q \cup E$, respectively, we obtain an equivalent formulation, customary called the *master program*

$$\begin{aligned} \min \quad & \sum_{q\in Q} c_q\lambda_q + \sum_{e\in E} c_e\lambda_e & & \\ \text{subject to} \quad & \sum_{q\in Q} \boldsymbol{a}_q\lambda_q + \sum_{e\in E} \boldsymbol{a}_e\lambda_e & \leq & \ \boldsymbol{b} \\ & \sum_{q\in Q} \lambda_q & = & \ 1 \\ & \boldsymbol{\lambda} & \geq & \ 0\ , \end{aligned}$$

typically having an astronomically large number $|Q| + |E| \gg n$ of variables, but possibly having substantially fewer rows than (CF). Although the original and the master program are equivalent

in that they give the same optimal objective function value, one has to point out that the respective polyhedra are not *combinatorially* equivalent (*see* e.g., ADLER & ÜLKÜCÜ 1973, NAZARETH 1987). The most important consequence is that the correspondence between solutions is not one to one. Because of its rôle in the representation (2.1), the equation $\sum_{q\in Q}\lambda_q = 1$ is often referred to as *convexity constraint*. The monographs of CHVÁTAL (1983) and LASDON (1970), among others, contain a thorough and detailed introduction to the subject.

Disregarding for a moment the large size of the reformulation in terms of the variables, the master program may be easier and/or faster to solve than the compact formulation, owing e.g., to the reduced size of the basis matrix when the simplex method is used. On the other hand, when (CF) is the linear programming relaxation of an integer program, nothing is gained with respect to the quality of the obtained lower bound on the corresponding integer optimal objective function value, c.f. the example in Figures 2.2 and 2.3. To put it more explicitly, the reformulation of a linear relaxation itself is of no value in a linear programming based branch-and-bound algorithm to solve the associated integer program. As we will see in Section 2.6, adding cutting planes to the master program in order to strengthen the relaxation is a problematic issue as well in column generation. Until recently, the latter was considered rather an *alternative* than a complementary reformulation of (CF). Nevertheless, the decomposition principle *is* commonly used in integer programming, and there with remarkable success. This is outlined next.

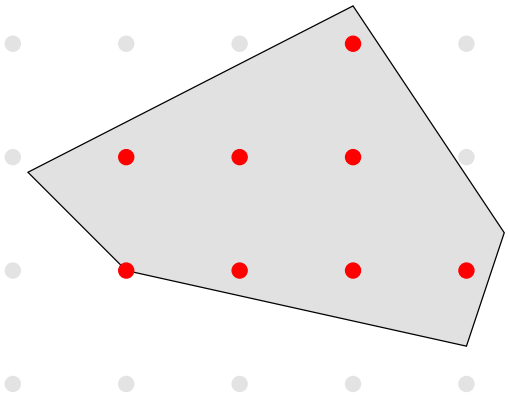

Figure 2.2: An example feasible region S

2.1.2 Decomposition of Integer Programs

Throughout our discussion we will only consider pure integer programs. The argumentation indeed carries over to the mixed integer case as well. Let $S = \{\boldsymbol{x} \in \mathbb{Z}^n \mid D\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{x} \geq \boldsymbol{0}\}$. The traditional presentation of decomposing an integer program again relies on the MINKOWSKI-WEYL theorem. The single difference is that the change of the polyhedral representation is applied to a different polyhedron, *viz.* $\mathrm{conv}(S)$, the convex hull of the discrete set S. VANDERBECK (1994, 1995) accounts for this state of affairs by calling this approach *convexification*. As Figure 2.4 suggests, however, the multipliers in (2.1) need not be integral. Extra arrangements to enforce integrality of the resulting master program are to be made in this case. HOLM & TIND (1988), for instance, require in their general framework $\sum_{q\in Q}\boldsymbol{p}_q\lambda_q + \sum_{e\in E}\boldsymbol{p}_e\lambda_e$ to be integer. Strictly speaking, this concept is still one of *linear* not integer programming. Imposing integrality on the multipliers, i.e., the variables of the master program, would not lead to an equivalent integer programming reformulation of the original problem, since the optimum integer solution of (CF) may be an interior point of $\mathrm{conv}(S)$. This circumstance leads VANDERBECK to an alter-

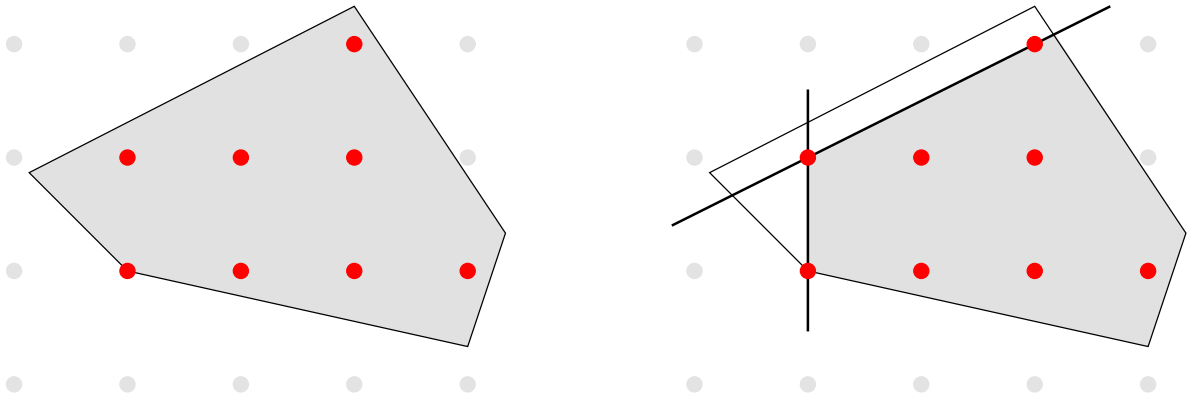

Figure 2.3: Effect of the reformulation in a linear programming framework. Shown is the result of the extensive reformulation in terms of affecting the "shape" of the feasible region S in Figure 2.2. Let (CF) be an integer program, i.e., $S = \{\boldsymbol{x} \in \mathbb{Z}^n \mid D\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{x} \geq \boldsymbol{0}\}$, and DANTZIG-WOLFE decomposition be applied to its linear programming relaxation. This merely alters the polyhedral representation of the linear relaxation of S, as shown on the left. That is, there is no change in the "shape" and the quality of the lower bound provided by the LP relaxation does not improve. Adding cutting planes—as depicted on the right—is a common remedy and is briefly discussed later. These cuts are not applied to the master program but to S. Note the difference!

native decomposition approach, for which he introduces the notion of integer *discretization*, *see* also VANDERBECK (2000). His reformulation of a compact integer program indeed takes the step to a master integer program, yielding a true integer analogue to the decomposition principle; *see* also JOHNSON (1989) for a similar presentation. Despite its similarity to the content of the previous subsection we detail the necessary steps, owing to its importance for the solution methodology adopted in this thesis. The development is based on the following result (*see* NEMHAUSER & WOLSEY 1988).

Theorem 2.1 (Finite Integral Generation of Integer Points in a Polyhedron)
Let $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid D\boldsymbol{x} \leq \boldsymbol{d}, \boldsymbol{x} \geq \boldsymbol{0}\} \neq \emptyset$ with $(D, \boldsymbol{d}) \in \mathbb{Q}^{m \times (n+1)}$ and $S = P \cap \mathbb{Z}^n$. Then there exists a finite set of integer points $\{\boldsymbol{p}_q\}_{q \in Q} \subseteq S$ and a finite set of integer rays $\{\boldsymbol{p}_e\}_{e \in E}$ of P such that

$$S = \left\{ \boldsymbol{x} \in \mathbb{R}^n_+ \mid \boldsymbol{x} = \sum_{q \in Q} \boldsymbol{p}_q \lambda_q + \sum_{e \in E} \boldsymbol{p}_e \lambda_e,\ \sum_{q \in Q} \lambda_q = 1,\ \boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|+|E|} \right\} .$$

Although this theorem enables us to fully parallel the threat of argumentation as above, we assume from now on for the remainder of this chapter that S is bounded and non-empty for the purpose of notational convenience, losing some generality but no practical relevance in doing so. Note, that this implies that $\mathrm{conv}(S)$ is a pointed polytope, and the set $\{\boldsymbol{p}_q\}_{q \in Q}$ coincides with S, making the theorem's statement obvious, since

$$\boldsymbol{x} \in S \iff \boldsymbol{x} = \sum_{q \in Q} \boldsymbol{p}_q \lambda_q,\ \sum_{q \in Q} \lambda_q = 1,\ \boldsymbol{\lambda} \in \{0,1\}^{|Q|} \quad . \tag{2.2}$$

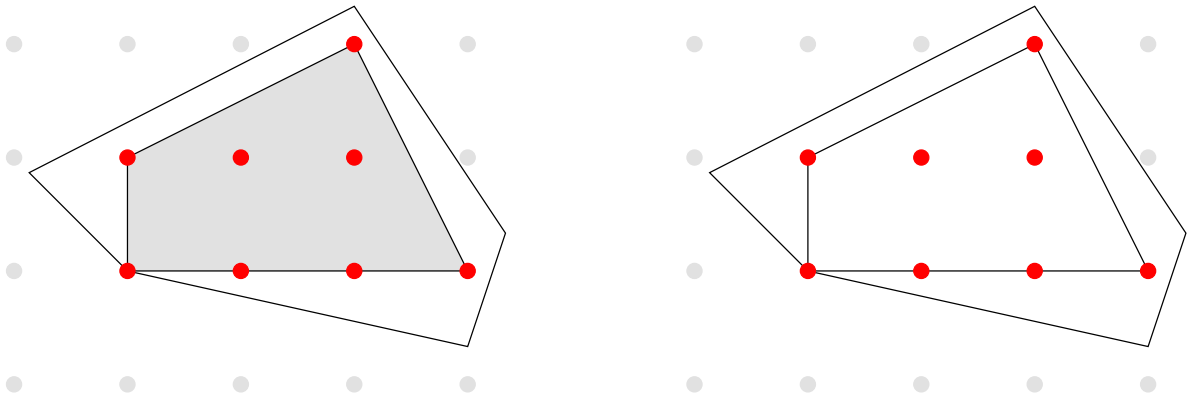

Figure 2.4: Towards a reformulation as an integer program. Depicted on the left is an approach to integer programming decomposition, called the *convexification* of $S = \{\boldsymbol{x} \in \mathbb{Z}^n \mid D\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{x} \geq \boldsymbol{0}\}$. If $\text{conv}(S)$ is not already an integral polyhedron, the bound provided by the linear programming relaxation of the reformulation is stronger than the one obtained from the original formulation (GEOFFRION 1974), explaining the popularity and effectiveness of this approach. However, convex combining the extreme points of $\text{conv}(S)$ in order to obtain interior integral points may involve fractional multipliers in (2.1). This leads to a direct reformulation of the integral points, i.e., S itself. This technique is called *discretization*, shown on the right. Both, convexification and discretization give the same bound obtained from the respective linear relaxation.

We indicate pitfalls to avoid in the unbounded case when appropriate. Substituting for $\boldsymbol{x}$ in the compact formulation as given by (2.2) we obtain a binary master program

$$\begin{aligned} \min \quad & \sum_{q \in Q} c_q \lambda_q & & \\ \text{subject to} \quad & \sum_{q \in Q} \boldsymbol{a}_q \lambda_q & \leq & \ \boldsymbol{b} \\ & \sum_{q \in Q} \lambda_q & = & \ 1 \\ & \lambda_q & \in & \ \{0,1\}, \quad q \in Q\ , \end{aligned} \tag{2.3}$$

where $c_q = c(\boldsymbol{p}_q)$ and $a_q = A\boldsymbol{p}_q$, $q \in Q$. The particularity to note here is the linearity of the objective function. The assumption of an additively separable cost function c is valid because $\boldsymbol{\lambda}$ is binary, and together with the convexity constraint the following holds

$$c(\boldsymbol{x}) = c\left(\sum_{q \in Q} \boldsymbol{p}_q \lambda_q\right) = c(\boldsymbol{p}_{q^\star}) = c_{q^\star} = \sum_{q \in Q} c_q \lambda_q$$

for precisely one $q^\star \in Q$. This transformation cannot be generalized to the unbounded case, since a solution $\boldsymbol{x}$ to the original formulation needs not have a unique corresponding solution $\boldsymbol{\lambda}$ to the master program as is the case when S is bounded. More precisely, mapping $\boldsymbol{\lambda}$ to $\boldsymbol{x}$ is not injective if and only if there exists $\bar{q} \in Q$ and $\bar{e} \in E$ such that $\boldsymbol{p}_{\bar{q}} + \boldsymbol{p}_{\bar{e}} \in S$.

Block Diagonal Structure

We now turn to the classical case of Figure 2.1 (a) which is typical to many applications as the one studied in this thesis. The decomposition can be carried further when the matrix D has block diagonal structure, i.e.,

$$D = \begin{pmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^K \end{pmatrix}, \qquad \boldsymbol{d} = \begin{pmatrix} \boldsymbol{d}^1 \\ \boldsymbol{d}^2 \\ \vdots \\ \boldsymbol{d}^K \end{pmatrix},$$

where $D^k \in \mathbb{Q}^{m_k \times n_k}$, $\boldsymbol{d}^k \in \mathbb{Q}^{m_k}$ with $\sum_{k=1}^{K} n_k = n$. Each set $S_k = \{\boldsymbol{x} \in \mathbb{Z}^{n_k} \mid D^k \boldsymbol{x} \leq \boldsymbol{d}^k,\ \boldsymbol{x} \geq \boldsymbol{0}\}$, $k = 1,\ldots,K$ independently gives rise to its own representation in the sense of Theorem 2.1. This segmentation is reflected by a superscript k to the entities c_q^k, $\boldsymbol{a}_q^k$, and λ_q^k for $q \in Q_k$, in our notation indicating the relation to the respective subsystem $k = 1,\ldots,K$. The binary master program constructed on the analogy of (2.3) reads

$$\begin{aligned} \min \quad & \sum_{k\in K} \sum_{q\in Q_k} c_q^k \lambda_q^k & & \\ \text{subject to} \quad & \sum_{k\in K} \sum_{q\in Q_k} \boldsymbol{a}_q^k \lambda_q^k & \leq & \ \boldsymbol{b} \\ & \sum_{q\in Q_k} \lambda_q^k & = & \ 1, \qquad k \in K \\ & \lambda_q^k & \in & \ \{0,1\}, \quad k \in K,\ q \in Q_k \ . \end{aligned} \tag{MP}$$

Note, that we abbreviated $k = 1,\ldots,K$ by $k \in K$. We will continue to use both notations. For later use we define $Q = \bigcup_{k\in K} Q_k$. If the null vector $\boldsymbol{x} \equiv \boldsymbol{0}$ is feasible for S in the original formulation at zero cost then it is sometimes left out of the formulation. The convexity constraints are then replaced by $\sum_{q\in Q_k} \lambda_q^k \leq 1$, $k \in K$.

Especially in the context of the next subsection, the elements $k \in K$ are called *resources*—*see* also the economic interpretation on page 39. Above we assumed them to be *distinct*, i.e., all subsystems are different. If the corresponding subsystems $S_1 = \cdots = S_K$ in the original formulation (CF) have identical characteristics, one speaks of *identical resources*. In this latter case, sometimes an *aggregated convexity constraint* is present in the master formulation

$$\begin{aligned} \min \quad & \sum_{q\in Q_1} c_q^1 \lambda_q^1 & & \\ \text{subject to} \quad & \sum_{q\in Q_1} \boldsymbol{a}_q^1 \lambda_q^1 & \leq & \ \boldsymbol{b} \\ & \sum_{q\in Q_1} \lambda_q^1 & = & \ K \\ & \lambda_q^1 & \in & \ \mathbb{Z}_+, \quad q \in Q_1 \ , \end{aligned}$$

where without loss of generality the first resource is used for notational reference. Following the naming convention of the above this results in the entities Q_1, c_q^1, $\boldsymbol{a}_q^1$, and λ_q^1, respectively. Note,

that we obtained a general integer program. Note also, that in principle any number $1 \leq i \leq K$ of subsystems can be aggregated; we display only the extreme cases here. *See* VANDERBECK (1995, 2000) for a discussion of the relationship between the respective domains of the variables in the various master formulations presented here.

Alternative Reformulations

We conclude our general discussion by pointing out that the presented scheme is not the only possible. According to the ideas of VANDERBECK (1994, 1995), the decomposition of integer programs amounts to replacing a subsystem S of the constraints of (CF) by a reformulation that possesses the *integrality property*. In fact, VALÉRIO DE CARVALHO (1999) formulates a master program for the cutting stock problem in terms of arc flow variables, i.e., each column corresponds to an arc in an appropriately defined network. Each path flow in that network gives rise to a valid cutting pattern for the cutting stock problem. He proves that the linear programming relaxation of this formulation can be derived from the classical extensive formulation of GILMORE & GOMORY (1961) by application of DANTZIG-WOLFE decomposition. The flow conservation constraints, of which there are exponentially many, are kept in the master program and are generated only as needed.

However, we would like to point out that the usefulness of the subproblem's integrality property is controversially discussed, *see* DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995). On the one hand, the subproblem can be solved via linear programming quite efficiently, or even by (ideally polynomial time) combinatorial algorithms. On the other hand, the lower bound obtained from solving the restricted master program does not improve upon the one obtained from solving the linear relaxation of the original formulation directly. Therefore, the integrality property may be undesirable, when the integrality gap is large. GEOFFRION (1974) comes to the very same conclusion for LAGRANGIAN relaxation, c.f. Subsection 2.3.2.

2.1.3 Set Partitioning and Set Covering Problems

When the integer feasible points of the subsystem to be reformulated are contained in the unit hypercube, i.e., $S \subseteq [0,1]^n$, the convexification and the discretization point of view coincide, since every integer point of S is an extreme point of $\text{conv}(S)$ and vice versa. Here, we will specialize even further to *set partitioning* and *set covering problems* (*see* BALAS & PADBERG 1972, 1975, 1976). We accent this model approach because of its relevance to practical applications. It is by far the most frequently discussed in our context, and almost all of the combinatorial optimization models for large-scale applications we are aware of are of this or related type. Although we could stick to the above presentation we introduce the concept in a slightly different way.

Denote by $\mathcal{R}$ a *ground set* of m elements, say $\{1,\ldots,m\}$. A subset $R \subseteq \mathcal{R}$ is encoded by an incidence vector $\boldsymbol{x}_R \in \{0,1\}^m$, the r^{th} component x_{rR} of which equals one if $r \in R$ and zero otherwise. It is convenient to identify a subset with the associate incidence vector, and we will interchangeably use both concepts for the same entity. We define a cost function $c : \{0,1\}^m \to \mathbb{R}$

on the powerset of $\mathcal{R}$. The *set partitioning problem* is to find a minimum cost selection of subsets of $\mathcal{R}$, such that each element of the ground set is contained in precisely one such set. To express set membership, we also say that an element is *covered* by the respective set. Reasonably, not every subset of $\mathcal{R}$ will be feasible for the underlying application. Assuming that we can represent the feasibility of subsets in terms of a polyhedral description $S = \{x \in \{0,1\}^m \mid D\boldsymbol{x} \leq \boldsymbol{d}\}$, the problem can be stated intuitively as

$$\begin{array}{rrcl} \min & \sum_{R \in X} c(\boldsymbol{x}_R) & & \\ \text{subject to} & \sum_{R \in X} \boldsymbol{x}_R & = & \mathbf{1} \\ & X & \subseteq & S \ , \end{array}$$

which, in a sense, gives a compact (but impractical!) formulation. As before, we convert the implicit information of feasibility into an explicit information collected in the constraint matrix of an appropriate reformulation.[3] The trivial "transformation" is $\boldsymbol{a}_s = \boldsymbol{x}_s$, and $c_s = c(\boldsymbol{x}_s)$, $s \in S$, respectively, and we obtain

$$\begin{array}{rrcl} \min & \sum_{s \in S} c_s \lambda_s & & \\ \text{subject to} & \sum_{s \in S} \boldsymbol{a}_s \lambda_s & = & \mathbf{1} \\ & \lambda_s & \in & \{0,1\}, \quad s \in S \ . \end{array} \tag{SP}$$

Here the binary variable λ_s indicates whether $s \in S$ is selected or not. For set partitioning problems it is often natural to think of having K different *resources*, as defined above, each of which may "contribute" at most one subset out of the respective S_k to the partition. Appropriate convexity constraints must be added to the above formulation.

Set covering problems require each element of the ground set to be covered at least once, thus equality in the *partitioning constraints* is replaced by "greater than or equal." VANDERBECK (1994) considers generalized set partitioning models, where the right hand side is not restricted to be the vector of all ones, and the variables may assume arbitrary integer values. Note, that in this case convexification is different from discretization.

2.2 Methodology Outline

In the previous section we have introduced the theoretical principles for different reformulations of a subsystem of constraints of a linear or integer program. However, an explanation of how to practically handle the resulting large models is still owing. Solving the linear master[4] directly, i.e., by means of the straight forward application of, say, the simplex method is definitively out

[3] It is interesting that in the decomposition context the converse is also a legitimate viewpoint: We convert the explicit information $S = \{x \in \{0,1\}^m \mid D\boldsymbol{x} \leq \boldsymbol{d}\}$ of feasibility into an implicit information $s \in S$.

[4] We refer to *any* extensive formulation as master program, regardless whether it is the result of a decomposition or not. The label (MP) is used throughout for this more general reference.

of reach. Even if memory requirements could be satisfied, excessive run times would render this solution approach impracticable due to the costly *explicit* search for a non-basic variable in each iteration—out of an elusive number of candidates—to price out and enter the basis. Besides this practical reason for not working with the master problem as it stands, support for this decision also comes on more theoretical grounds:

① When (MP) actually results from one of the above decomposition schemes the respective sets of (extreme) points and extreme rays are not readily available. No efficient procedure for converting the polyhedral description is known, not even from the widened standpoint of the computational complexity theory for enumeration problems. To the best of our knowledge "efficient" algorithms were proposed only for the case of simple polyhedra (AVIS & FUKUDA 1992) and 0/1 polytopes (BUSSIECK & LÜBBECKE 1998) but even these exceptions do not allow for explicitly writing down an extensive master program.

② The vast majority of columns will have their associated variable at level zero in any basic solution anyway. Thus, in principle, only a tiny fraction of the whole information is needed at a time in the simplex method.

③ Although, on average, the number of simplex pivots is polynomial in the number of constraints *and* variables of a linear program, respectively (BORGWARDT 1982), m to $3m$ pivot steps required is a commonly used rule of thumb (CHVÁTAL 1983). From this one might tend not to expect a significant growth in the number of iterations necessary to solve the extensive formulation. However, *see* CHVÁTAL (1983) for a different point of view.

We will explain now that, for instance, the generic[5] revised simplex method, *see* e.g., the book of CHVÁTAL (1983), is perfectly suited for carrying out the pricing *implicitly* without having to have the whole coefficient matrix at hand. To begin with, recall that in the revised simplex method all data required per iteration is calculated directly from the original data—in contrast to e.g., the tableau method which modifies the *whole* input data from iteration to iteration. What is more, only the pricing step needs access to non-basic columns of the coefficient matrix when it comes to computing the reduced cost coefficients. Let the master program under consideration have the form of the linear relaxation of (MP) on page 32, which we denote by (MP′). We will stick to the following notational convention: For all (integer) problems (P) introduced we will use the primed label (P′) for referring to the linear relaxation of (P). When (P) already constitutes a linear program, both labels denote the very same problem. Let $z_{\mathrm{P}}^{\star}$ denote the optimal objective function value of problem (P). Arranging the dual variables according to the two types of constraints (linking and convexity), the dual of the relaxed master program is

$$\begin{array}{rrcl} \max & \boldsymbol{u}^{\mathsf{T}}\boldsymbol{b}+\boldsymbol{v}^{\mathsf{T}}\mathbf{1} & & \\ \text{subject to} & \boldsymbol{u}^{\mathsf{T}}\boldsymbol{a}_q^k+v_k & \leq & c_q^k, \quad k\in K,\ q\in Q_k \\ & \boldsymbol{u} & \leq & \mathbf{0} \end{array} \tag{DMP′}$$

[5] This is to remind us of the degree of freedom when implementing *the* revised simplex method, e.g., the different choices of basis factorizations. Our argumentation is not particular to a specific implementation, and we will drop the *generic* henceforth.

from which we immediately read the condition for dual feasibility, or equivalently, primal optimality. We refer to the feasible region of (DMP′) as *dual polyhedron*. However, this information is not of seizable use either as we have made an effort to point out that it is not only unlikely but also undesirable to have the entire matrix $(\boldsymbol{a}_q^k)_{k\in K, q\in Q}$ at our disposal.

Restricted Master Program

Instead, we will work with a *manageable subset* $Q' \subseteq Q$ of columns, at the worst starting with a set which contains only one primal feasible basis for (MP′). Although obtaining this initial set constitutes a problem in its own right, c.f. Subsection 2.3.3, let us assume for the moment that we are provided with such a column set. As customary in the literature, the master program with columns omitted is called *restricted*. Without surprise it reads

$$\begin{aligned} \min \quad & \sum_{k\in K}\sum_{q\in Q'_k} c_q^k \lambda_q^k & & \\ \text{subject to} \quad & \sum_{k\in K}\sum_{q\in Q'_k} a_{qi}^k \lambda_q^k & \leq & b_i, \quad i \in \{1,\dots,m\} \\ & \sum_{q\in Q'_k} \lambda_q^k & = & 1, \quad k \in K \\ & \lambda_q^k & \geq & 0, \quad k \in K,\ q \in Q'_k \ . \end{aligned} \tag{RMP′}$$

where $Q'_k \subseteq Q_k$, $k = 1,\dots,K$, and again $Q' = \bigcup_{k\in K} Q'_k$. Analogously, we define the dual of the restricted master program labeled by (DRMP). To stress the distinction to their restricted versions we will sometimes refer to (MP) and (DMP) as *full (dual) master program*, respectively.

Since in each iteration of the simplex method exactly one basic column is exchanged for one non-basic column, *on-the-fly* generation of columns to enter the basis is an appealing idea. Its realization goes as follows. Associated with a primal optimal solution $\boldsymbol{\lambda}^\star \in \mathbb{R}^{|Q'|}$ to (RMP′) is a dual optimal solution $(\boldsymbol{u}^\star, \boldsymbol{v}^\star)^\mathsf{T} \in \mathbb{R}^{m+K}$. Note again, that the optimization to obtain this solution is carried out having a (very small) subset of columns at hand but checking optimality of $\boldsymbol{\lambda}^\star$ with respect to the full program (MP′) requires testing non-negativity of *all* reduced cost coefficients. This amounts to solving the *pricing subproblem*

$$z_{\mathrm{PP}}^\star := \min\left\{ c_q^k - \boldsymbol{u}^{\star\mathsf{T}} \boldsymbol{a}_q^k - v_k^\star \mid k \in K,\ q \in Q_k \right\} \ . \tag{PP}$$

If $z_{\mathrm{PP}}^\star \geq 0$, *no* reduced cost coefficient has negative value and $\boldsymbol{\lambda}^\star$ (embedded in $\mathbb{R}^{|Q|}$ by setting the components in $Q \setminus Q'$ to zero) optimally solves (MP′) as well. Otherwise, i.e., $z_{\mathrm{PP}}^\star < 0$, the index q_z for which the minimum in (PP) is attained is adjoined to the respective Q'_k, thus enlarging (RMP′) by the column $\boldsymbol{a}_{q_z}^k$ (extended by a unit coefficient in the corresponding convexity constraint) at cost $c_{q_z}^k$. This completes the iteration and the enlarged restricted master program is re-optimized. For its rôle in the algorithm (PP) is also called the *generation problem*, or the *column generator*. The procedure stops as soon as no column with negative reduced cost coefficient is found.

This is the idea of column generation in terms of the simplex method. Algorithm 2.2 summarizes these general considerations. Of course, one crucial part here is the pricing problem itself which inherits the difficulty of searching virtually all non-basic columns and still we need to supply an explanation how to cope with this. One might get the impression that we are only delaying the trouble, but remember that we are provided with a valuable information about how possible columns "look like," namely the polyhedral description defined by the reformulated subsystem in the decomposition approach. In Section 2.4 we will see how to solve (PP) using this knowledge. With this background finally, the use of the notion *decomposition* becomes justified.

Algorithm 2.2 Generic Linear Programming Column Generation

Provide feasible basis for restricted master program // c.f. Subsection 2.3.3
while a column $\binom{c}{a}$ with negative reduced cost exists **do** // c.f. Section 2.4
  Adjoin $\binom{c}{a}$ to restricted master program
  Re-optimize // c.f. Section 2.3
end while

Growing Significance

A search for "column generation" in the global index of *Zentralblatt MATH*[6] (restricted to the 1991 Mathematics Subject Classifications 90Bxx and 90Cxx) returns 202 entries, c.f. the publication figures as per margin. We emphasize, however, that some important contributions do not even mention this notion. There is common consent that the underlying theoretical principle, DANTZIG-WOLFE decomposition, applied to linear programs usually performs poorly, c.f. Subsection 2.5. Nevertheless, we witness an impressive growth of papers reporting on the successful application of column generation techniques, particularly for (mixed) integer programming problems. Table 2.1 on the following page lists some of the attacked practical and theoretical problems, especially from the fruitful period of the past five years.

years	articles
– 1979	10
1980 – 1989	45
1990 – 2000	148

Figures as of May 15, 2001

Remark. The term *generalized linear programming* refers to the situation that a linear program is not statically stated, but defined in such a way that the data—especially the constraint matrix—is dynamically drawn from a well defined set. Since this is precisely the way column generation works these two names are synonymously used, the former being predominantly found in the early literature and probably obsolete today.

[6] Electronically published by Springer-Verlag. The relevant URL is `http://www.emis.de/ZMATH/`

Author(s)	Application(s)
AGARWAL, MATHUR & SALKIN (1989)	single-depot VRP
ANBIL, FORREST & PULLEYBLANK (1998)	airline crew pairing
BARNHART, BOLAND, CLARKE, JOHNSON, NEMHAUSER & SHENOI (1998)	fleet assignment and aircraft routing
BARNHART, HANE & VANCE (1997, 1998)	(origin-destination) integer multicommodity flows
BARNHART & SCHNEUR (1996)	air network design for express shipment service
BOURJOLLY, LAPORTE & MERCURE (1997)	maximum stable set problem
CRAINIC & ROUSSEAU (1987)	airline crew pairing
CRAMA & OERLEMANS (1994)	job grouping for flexible manufacturing systems
DESAULNIERS, DESROSIERS, DUMAS, SOLOMON & SOUMIS (1997)	daily aircraft routing and scheduling
DESAULNIERS, DESROSIERS & SOLOMON (1999)	vehicle routing and crew scheduling
DESROCHERS, DESROSIERS & SOLOMON (1992)	single-depot VRPTW
DESROCHERS & SOUMIS (1989)	urban transit crew scheduling
DESROSIERS, SOUMIS & DESROCHERS (1984)	*m*-TSPTW
EBEN-CHAIME, TOVEY & AMMONS (1996)	grouping and packaging of electronic circuits
ERDMANN, NOLTE, NOLTEMEIER & SCHRADER (1999)	airline schedule generation
HANSEN, JAUMARD & POGGI DE ARAGÃO (1998)	probabilistic maximum satisfiability problem
HURKENS, DE JONG & CHEN (1997)	cutting strips problem (a variant of cutting stock)
IOACHIM, DESROSIERS, SOUMIS & BÉLANGER (1999)	fleet assignment and routing with schedule synchronization
JOHNSON, MEHROTRA & NEMHAUSER (1993)	minimum cut clustering
LÖBEL (1997, 1998)	multiple-depot vehicle scheduling
MEHROTRA & TRICK (1996)	graph coloring
PARK, KANG & PARK (1996)	bandwidth packing in telecommunication networks
RIBEIRO, MINOUX & PENNA (1989)	traffic assignment in satellite communication systems
RIBEIRO & SOUMIS (1994)	multiple-depot vehicle scheduling
SANKARAN (1995)	course registration at a business school
SAVELSBERGH (1997)	generalized assignment problem
SOL (1994)	*m*-PDPTW
VALÉRIO DE CARVALHO (1999, 2000)	bin packing and cutting stock problems
VANCE, ATAMTÜRK, BARNHART, GELMAN, JOHNSON, KRISHNA *et al.* (1997)	airline crew pairing
VANCE (1998)	one dimensional cutting stock problem
VANCE, BARNHART, JOHNSON & NEMHAUSER (1994)	binary cutting stock problems
VANDERBECK (1994)	network design with split assignments graph partitioning e.g., in VLSI, compiler design single-machine multi-item lot-sizing
VANDERBECK (1999)	bin packing and cutting stock problems

Table 2.1: Selected applications of integer programming column generation. This list is not complete by far and shows only some of the more recent practical and combinatorial problems attacked.

2.3 The Restricted Master Program

At all times, the restricted master program represents all current problem information gathered from subproblem solutions, i.e., a subset of columns of the coefficient matrix of a linear program which proved to be useful to achieve progress in terms of the objective function value. From a technical point of view, the purpose of the restricted master program is twofold. Firstly, to combine columns/variables in an appropriate way in order to obtain a primal feasible solution, and secondly, to provide dual multipliers to be transferred to the subproblem in order to promisingly extend the current information. For its rôle in this two-level algorithm, the restricted master program is often referred to as *coordination problem*, c.f. Figure 2.5.

Economic Interpretation

The decomposition principle has a customary interpretation as *decentralized planning without complete information at the center*. Consider different production facilities, or *branches*, of an industry which are independent of one another in the sense that none knows about the amount the others produce. Despite this ignorance, however, the branches each try to make (best) use of a set of common scarce resources, and make appropriate production proposals to a central, superordinated resource manager. This manager does not know about what particular constraints lead the branches to their decisions, but has a global overview of the overall resource usage, and on the respective net profit contributed by each of the branches; moreover, he or she is entitled to combine (fractions of) the respective proposals to a tentative production plan (obeying resource constraints), but cannot directly dictate decisions to the branches. Instead, the manager introduces *shadow prices* for the resources, which represent precisely that price, each branch *would* have to pay for each resource when realizing the assigned fraction of its production proposal in order to break even. In what regards a global production plan, a new proposal by a branch pays off if and only if the net profit brought in by the proposal exceeds the cost for its resource consumption measured by the shadow prices. LASDON (1970) and CHVÁTAL (1983) give excellent expositions also in mathematical terms.

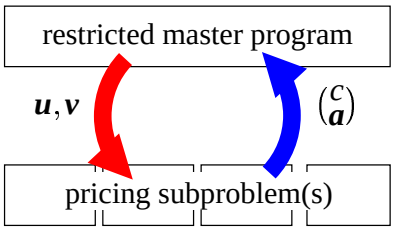

Figure 2.5: Information flow between master program and subproblem(s)

2.3.1 Dual Variables

Primal methods, like column generation, maintain primal feasibility and work towards dual feasibility. It is therefore only natural to monitor the dual solution in the course of the algorithm. In our opinion, the dual point of view reveals most valuable insight into the algorithm's functioning. We elaborate on this perspective in more detail throughout the whole chapter.

Extreme Point Versus Interior Solutions

A dual solution obtained from the restricted master program needs not be unique, i.e., it may be degenerate. That is, there exists an optimal face of the dual polyhedron with dimension greater than zero. Actually, if the primal is degenerate—combinatorial optimization problems typically give rise to highly degenerate linear relaxations—this is likely to happen. This is significant inasmuch the dual solution directly influences the selection of new columns. Since in the simplex method a primal basic solution is associated with a dual basic solution we obtain an extreme point of the face in question. Sometimes the argument is brought forward that such an extreme point is not a good *representative* of the multitude of dual solutions obtainable.

BIXBY, GREGORY, LUSTIG, MARSTEN & SHANNO (1992) argue that solving the restricted master program by the simplex method leads to an optimal basis essentially chosen at random, whereas the application of an interior point method produces a solution in the relative interior of the optimal face. Therefore, e.g., *analytic* and *volumetric centers* have been proposed by several authors, and one such proposal will be sketched in the next subsection. A related approach, which stays within the linear programming framework, is considered by VANDERBECK (1994). The restricted master program is first solved to obtain the optimal objective function value, and a second time with a different objective function, *viz.* maximizing the sum of auxiliary variables which bound the dual variable values on the optimal face from below. However, the such calculated *pseudo-central costs* are detrimental to the subproblem performance.

The opinions found in the literature diverge about whether to use extreme point dual solutions or not. VANDERBECK (1994) opts for their use *because* of their "random" nature, which possibly results in different, even complementary kinds of columns. Not least a smaller computational burden is an advantage as well. On the contrary, ANBIL, FORREST & PULLEYBLANK (1998) deny the efficiency of extreme point solutions in their column generation approach to the airline crew pairing problem, because they "tend to be very sparse."

2.3.2 Lagrangian Duality

A well-known alternative way of getting lower bounds on the optimal objective function value of an integer program is provided by LAGRANGIAN duality. Again, let (CF) represent the compact formulation of an integer program. We call

$$\mathcal{L}(\boldsymbol{u}) := \begin{array}{ll} \min & c(\boldsymbol{x}) - \boldsymbol{u}^{\mathsf{T}}(\boldsymbol{b} - A\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in S \end{array} , \tag{2.4}$$

the *Lagrangian relaxation* of (CF) with respect to $A\boldsymbol{x} \leq \boldsymbol{b}$ and LAGRANGE multipliers $\boldsymbol{u}$. This clearly gives a lower bound on $z^{\star}_{\mathrm{CF}}$ for any $\boldsymbol{u} \geq 0$. The natural question for the best such bound is asked by the LAGRANGIAN dual problem

$$\max\{\mathcal{L}(\boldsymbol{u}) \mid \boldsymbol{u} \geq \boldsymbol{0}\} \quad . \tag{2.5}$$

The LAGRANGIAN dual (2.5) and the linearly relaxed master program (MP′) derived from (CF) in fact provide the very same bound, as was established by GEOFFRION (1974). He also

made clear that the pricing problem in column generation is precisely of the form (2.4). Moreover, the LAGRANGE multipliers $\boldsymbol{u}$ correspond to the dual variables with the same notation. In view of furnishing values for $\boldsymbol{u}$—the aim of this section—column generation is considered as a primal method, and LAGRANGIAN relaxation as the equivalent dual method to be used on the same decomposable structures (DESAULNIERS, DESROSIERS & SOLOMON 1999). When there is choice,[7] preference should be given to the method which more effectively yields an optimal $\boldsymbol{u}$.

Subgradient algorithms (*see* e.g., WOLSEY 1998) have so far been the method of choice for solving the LAGRANGIAN dual. One objection against using this particularly simple method is that it exploits only *local* information for the iterative update of the dual variables. On the contrary, solving a restricted master program is a more elaborate update strategy which makes use of *all* the information gathered during the solution process. Only, the partly occurring large linear programs could not be solved until recently. Undoubtedly, computational progress here helped column generation to its merited popularity. BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998) leave it open to future research whether LAGRANGIAN relaxation in connection with more advanced (nonlinear) alternatives to subgradient algorithms are able to outperform the column generation approach in general.

2.3.3 Initial Basis of the Restricted Master Program

When we use simplex method to solve the restricted master program an initial basis is required. CHVÁTAL (1983) details how a primal feasible solution can be derived by means of a *first phase* procedure. Although this is in perfect analogy with what is used to start the simplex method, more tailored proposals have been made for constructing the first basis. Actually, in view of the above, the initialization is of crucial importance for the whole procedure.

Artificial Start

An initialization with $Q'_k = \emptyset$, $k \in K$, is possible by introduction of *artificial variables* $\boldsymbol{y} \in \mathbb{R}^{m+K}_+$, one for each constraint of the restricted master program. Usage of these variables is *penalized* via a large constant $M > \max_{k\in K, q\in Q_k} c^k_q$, i.e.,

$$
\begin{array}{rrcrcl}
\min & \displaystyle\sum_{k\in K}\sum_{q\in Q'_k} c^k_q \lambda^k_q & + & \displaystyle\sum_{i=1}^{m} M y_i & & \\
\text{subject to} & \displaystyle\sum_{k\in K}\sum_{q\in Q'_k} a^k_{qi} \lambda^k_q & + & y_i & \leq & b_i, \quad i \in \{1,\ldots,m\} \\
& \displaystyle\sum_{q\in Q'_k} \lambda^k_q & + & y_{m+k} & = & 1, \quad k \in \{1,\ldots,K\} \\
& \lambda^k_q & & & \geq & 0, \quad k \in \{1,\ldots,K\},\ q \in Q'_k \\
& & & y_i & \geq & 0, \quad i \in \{1,\ldots,m+K\} \ .
\end{array}
\tag{2.6}
$$

[7] For too large numbers of dual variables to adjust, LAGRANGIAN relaxation is the only practicable alternative.

The unit matrix corresponding to the artificial columns constitutes a feasible basis matrix. The penalty cost ensure that artificial variables are driven out of the basis. As soon as this happens, a feasible solution to the original (RMP$'$) is found—assuming that a feasible solution exists. Note, that variables $y_{m+1},\ldots,y_{m+K}$ can be left out of the formulation, when the convexity constraints are in their *less-or-equal* form. Experimentation with an actual choice for M is advisable, since a smaller M gives a tighter initial bound on the associated dual variables, as we have just explained. Since (2.6) uses a combination of original and artificial objective functions, we limit the generation of columns that contribute to feasibility only, but whose cost are too large to be of any value in an optimal solution.

Refinements and Alternatives

VANDERBECK (1994) enlarges in detail on initialization, especially in the context of embedding column generation in a branch-and-bound algorithm, and summarizes that "with an appropriate initial set of columns, one can get a good start in the column generation procedure." We might add, that the negation of this assertion holds as well.

In some applications, *see* e.g., the vehicle routing context of AGARWAL, MATHUR & SALKIN (1989), the unit basis may represent a feasible solution. Of course, in this case, (an approximation of) the respective actual cost coefficients should be used instead of M. The latter authors impose artificial upper bounds of the respective estimated optimal values on the dual variables by introduction of unit columns with appropriate cost. These estimates are obtained heuristically, exploiting good problem knowledge. The bounds are gradually relaxed until they are no longer binding. We also refer to the related discussion of the *stabilization approach* in Subsection 2.5.4. Similar techniques have been proposed by other authors as well, *see* e.g., VANDERBECK (1994).

An alternative initial basis may be produced by a primal heuristic, which is of course problem specific. Here, we would like to make the point, that a poorly chosen set of initial columns, e.g., provided by a bad heuristic, may simply lead the column generation algorithm astray. An initial heuristic solution, which does not resemble the structure of a possible optimal solution at all must then be interpreted as a misleading bound on an irrelevant linear combination of the dual variables. VANDERBECK (1994) observes that even an excellent initial integer solution may be detrimental to solving a linear program by column generation. On the other hand, VALÉRIO DE CARVALHO (2000) makes good experiences with bounds on meaningful linear combinations of dual variables, c.f. the discussion of dual cutting planes in Subsection 2.4.1.

Let us finally mention that e.g., ANBIL, FORREST & PULLEYBLANK (1998) suggest a warm start from solutions obtained in earlier, similar runs, if available. This again reflects the need for *good* (primal or dual) solutions to the *linear* program.

2.3.4 Alternatives to the Simplex Method

Clearly, the restricted master program constitutes a linear program, and as a matter of course, the simplex method is the most obvious choice for its solution. LASDON (1970) comments on the

suitability of primal, dual, and primal-dual variants. However, particular approaches have been tailored to capture specific characteristics of linear programs encountered in column generation. In this subsection we briefly review selected proposals. For detailed presentations we refer to the respective citations given.

Sprint Method

In certain applications, the sheer size of the restricted master program may prohibit the use of the simplex method, simply for performance reasons. The commonness of the linear programs to be solved is that they typically have relatively few rows and a comparably large number of columns. Exploiting this fact, the main idea of the *Sprint method* (introduced by FORREST 1989) is to solve a large linear program by sequentially solving considerably smaller parts. The procedure can be interpreted as a column generation scheme by itself. We consider for its presentation a restriction of the familiar restricted master program, i.e.,

$$\begin{array}{rrcl} \min & \boldsymbol{c}_W^{\mathsf{T}}\boldsymbol{\lambda}_W & & \\ \text{subject to} & A_W\boldsymbol{\lambda}_W & = & \boldsymbol{b} \\ & \boldsymbol{\lambda}_W & \geq & \mathbf{0}\ , \end{array} \tag{2.7}$$

with $W \subseteq Q'$ a *working set* of columns. From (2.7) we obtain a dual optimal solution, which we use to price out *all* columns in $Q' \setminus W$. We would retain in W the columns which correspond to the optimal basic solution, and adjoin, according to their reduced cost coefficients, the most promising yet unused columns, as well as a small random selection of the remaining columns. We iterate until all columns have been considered, and finally, only once, the full linear program, i.e., $W = Q'$, is solved. ANBIL, FORREST & PULLEYBLANK (1998) and CHU, GELMAN & JOHNSON (1997) report this approach to effectively solve linear programs which arise from set partitioning problems with more than five million variables. The approach is called *sifting* by BIXBY, GREGORY, LUSTIG, MARSTEN & SHANNO (1992) who solve the involved linear programs via a hybrid of an interior point method (which makes good progress initially) and the simplex method (which is fast in the end when only few columns are added).

BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998) relate the Sprint method to "column generation for non decomposable models." That is, problems e.g., defined on graphs are first solved on a *good* subset of arcs, then the remaining arcs have to be priced out in order to prove optimality or to identify arcs to be included in the problem.

Volume Algorithm

Rather than computing an exact solution, an approximation may suffice. The *volume algorithm*, introduced by BARAHONA & ANBIL (2000), is an extension of subgradient algorithms, and rapidly produces primal as well as dual approximate solutions to a linear program. It is named after a new way of looking at linear programming duality, using volumes below the active faces to compute the dual variable values and the direction of movement. The fundament is the following.

Theorem 2.3 (Volume and Duality, BARAHONA & ANBIL 2000)
Consider the linear program $\max\{v \in \mathbb{R} \mid v + \boldsymbol{a}_q^\mathsf{T}\boldsymbol{u} \le c_q,\ q = 1,\dots,|Q'|\}$. *Let* $(v^\star, \boldsymbol{u}^\star)$ *be a primal optimal solution in which constraints* $1,\dots,m' \le |Q'|$ *are active. Let* $\overline{v} < v^\star$. *Further assume the polyhedron* $\{v \in \mathbb{R} \mid v + \boldsymbol{a}_q^\mathsf{T}\boldsymbol{u} \le c_q,\ q = 1,\dots,m',\ v \ge \overline{v}\}$ *be bounded. Define* γ_q, $q = 1,\dots,m'$, *as the volume between the face induced by* $v + \boldsymbol{a}_q^\mathsf{T}\boldsymbol{u} = c_q$ *and the hyperplane defined by* $v = \overline{v}$, *c.f. Figure2.6. Then, an optimal dual solution is induced by*

$$\lambda_q = \frac{\gamma_q}{\sum_{i=1}^{m'} \gamma_i},\ q = 1,\dots,m' \quad . \tag{2.8}$$

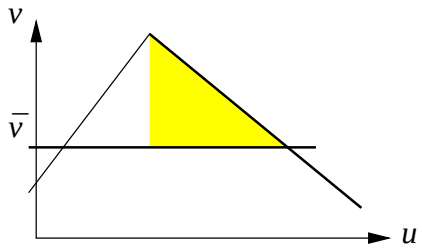

Figure 2.6: Shaded is the volume defined by a face and $v = \overline{v}$.

We chose a notation that reveals the theorem's applicability to (a slight reformulation of) the dual restricted master program (*see* also Subsection 2.3.2). An interesting interpretation is as follows. Call the pricing subproblem with a dual solution "in a neighborhood" (defined by $\overline{v}$) of an optimal dual solution. Then, (2.8) gives the probability λ_i that column $\boldsymbol{a}_i$ (which induces a face of the dual polyhedron) is generated. The subgradient method is modified in order to furnish estimates of these probabilities, i.e., an approximate primal solution. It must be stated, that primal feasibility may be mildly violated.

ANBIL, FORREST & PULLEYBLANK (1998) use the volume algorithm in alternation with the simplex method. The authors remark that the former produces dual solutions with a large number of variables non zero—a state of affairs they refer to as *highly non basic*; a quality which is claimed to accelerate column generation for reasons as discussed in Subsection 2.3.1. Promising computational experience is given for various combinatorial optimization problems, *see* e.g., BARAHONA & ANBIL (1998). The authors accredit to the volume algorithm straight forward implementation with small memory requirements, numerical stability, and fast convergence.

Dual Coordinate Search

Another approximation algorithm, but of a different kind, has been proposed for solving binary integer programs, i.e., $\min\{\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda} \mid A\boldsymbol{\lambda} = \boldsymbol{b},\ \boldsymbol{\lambda} \in \{0,1\}^{|Q'|}\}$, with $A \in \{0,1\}^{m \times |Q'|}$, and $\boldsymbol{b} \in \mathbb{Z}^m$. The motivation is to circumvent the detour of solving linear relaxations, but to solve an integer (restricted master) program directly. WEDELIN (1995) bases his coordinate search algorithm on the observation that an optimal integer solution to the LAGRANGIAN relaxation

$$\min_{0 \le \boldsymbol{\lambda} \le 1} \boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda} + \boldsymbol{u}^\mathsf{T}(\boldsymbol{b} - A\boldsymbol{\lambda})$$

optimally solves our binary program as well, and that such a solution is trivially deduced from the sign of the reduced cost coefficients $\overline{\boldsymbol{c}}^\mathsf{T} = \boldsymbol{c}^\mathsf{T} - \boldsymbol{u}^\mathsf{T}A$, *viz.* $\lambda_j = 0$ if $\overline{c}_j > 0$ and $\lambda_j = 1$ if $\overline{c}_j < 0$. In the case $\overline{c}_j = 0$, any, in particular fractional, $\lambda_j \in [0,1]$ is allowed, hence this is to be avoided.

The idea is now to construct a dual solution $\boldsymbol{u}$ such that $\bar{\boldsymbol{c}} \neq \boldsymbol{0}$. This is done by cyclically assigning values to the respective u_i, $i = 1, \ldots, m$, each time ensuring that the induced solution $\boldsymbol{\lambda}$ fulfills the i^{th} constraint in $A\boldsymbol{\lambda} = \boldsymbol{b}$. To be exact, define $\tilde{c}_{ij} := (\bar{c}_j + u_i) \cdot a_{ij}$, which gives for each i a set of reduced costs where the dual contribution, if any, of the i^{th} constraint is canceled. When the iteration comes to considering variable u_i, define c^- as the b_i^{th} smallest value among the $\tilde{c}_{ij}$, $j = 1, \ldots, |Q'|$, and c^+ as the $b_i + 1^{\text{st}}$ smallest. If it holds that $c^-, c^+ \neq 0$, the effect of setting $u_i := (c^- + c^+)/2$, and updating $\bar{\boldsymbol{c}}$ is that exactly b_i among the $\tilde{c}_{ij}$, $j = 1, \ldots, |Q'|$ are negative.

Still, it may (and will) happen, that zero reduced cost are produced, and no integer solution is found. The proposed remedy is a controlled perturbation of the original cost coefficients, thus resulting in an approximation. One variant of the algorithm can be understood in the linear programming framework. In fact, the coordinate search is shown to correspond to maximizing the LAGRANGIAN dual function. WEDELIN (1995) reports on high quality solutions for large scale set covering problems.

Analytic Center Cutting Plane Method

In our above discussion about *representative* dual solutions we saw that an interior point in the respective optimal face of the dual polyhedron may prove advantageous. However, even such a solution only refers to the *current* optimal linear combination of columns generated so far. Instead, and one step further, one could come up with dual variable values which represent, in a well defined sense, *all* information presently available. This is the idea of using *central prices*. In the context of DANTZIG-WOLFE decomposition it has been proposed by GOFFIN, HAURIE, VIAL & ZHU (1993).

Informally, we consider the logarithm of the product of slack variables in a reformulation of the dual restricted master program which ensures boundedness. This logarithm is well-defined on the interior of the dual polytope, is strictly concave, and tends to $-\infty$ towards the boundary. Therefore, a global maximum is attained on its domain. Under technical assumptions, the maximizer, i.e., the associated dual solution, is unique, and is called the *analytic center* of the dual polytope. It is this analytic center which is used as an alternative to extreme point dual solutions.

We do not describe how the analytic center is computed, but refer to the large body of literature which has developed, far beyond the scope of this chapter, c.f. the recent survey by GOFFIN & VIAL (1999). The problem gets solved by standard nonlinear techniques, provides a solution to the full dual master, hence a lower bound for the primal optimum; it allows a warm start from prior solutions, and produces stable dual variables (DESAULNIERS, DESROSIERS, IOACHIM, SOLOMON, SOUMIS & VILLENEUVE 1998), c.f. Subsection 2.5.4. An implementation is available for academic purposes at `http://ecolu-info.unige.ch/logilab/software/accpm/`.

Concluding Remarks

Currently, we cannot expect more general advice on what method works best, except for "try it!" Even worse, many more approaches to produce a dual solution to a linear program may come

to mind, each with their respective strengths and weaknesses. Our main goal in this section therefore was to collect some inspiration. Elaborate hybrids may evolve, *see* BORNDÖRFER & LÖBEL (2001), BIXBY, GREGORY, LUSTIG, MARSTEN & SHANNO (1992), ANBIL, FORREST & PULLEYBLANK (1998), and DESAULNIERS, DESROSIERS & SOLOMON (1999) for a terse suggestion. Of course, heuristics can be (and are) used to construct or improve dual variable values at any time in the algorithm.

Still, certain methods are better suited for particular purposes than others. For instance, in presence of primal degeneracy, the dual simplex often is to be preferred to the primal. Sometimes, however, a primal feasible solution needs to be maintained, e.g., for early termination, c.f. Subsection 2.5.3. The choice of method in general will also depend on how fast or how accurate a solution is needed, whether particular problem structures are present, and what implementation skills or solvers are available. One apparent trend is to refrain from insisting on optimality, and attack even larger problems, for which still a guaranteed approximation quality can be obtained.

2.4 The Pricing Problem

The *pricing step* in the simplex method is the task to *price out* the non-basic variables, i.e., to determine one with negative reduced cost coefficient (minimization assumed) which may enter the basis. In looking for such a column, we distinguish—although seemingly not customary in the literature—between *pricing schemes* and *pricing rules*, the former describing the (sub-)set of non-basic variables to consider, and the latter referring to the criterion according to which a column is selected from the chosen (sub-)set. DANTZIG's classical to choose among *all* columns the one with *most negative* reduced cost coefficient is an example for such a scheme/rule pair. Various schemes are proposed in the literature like *full*, *partial*, or *multiple pricing*; BIXBY (2000*b*) gives a most recent overview. In this sense, column generation is a pricing scheme for large-scale linear programs. To each *standard* pricing rule there exists a column generation *sibling*: Instead of pricing out non-basic variables by enumeration, e.g., the most negative reduced cost coefficient is found by solving the optimization problem

$$z^\star := \min \left\{ c_q^k - \boldsymbol{u}^\mathsf{T} \boldsymbol{a}_q^k - v_k \mid k \in K,\, q \in Q_k \right\} \quad . \tag{PP}$$

Note, that we dropped the superscript star from the dual optimal solution $(\boldsymbol{u}, \boldsymbol{v})^\mathsf{T}$ of (RMP′) for notational convenience. In general, using this particular pricing scheme is obviously more costly than using standard pricing schemes. Despite the fact that the former cannot compete with the latter in terms of computational efforts, its use is justified by extending the range of applicability of the simplex method to problem sizes impracticable to standard implementations. VANDERBECK (1994) reports on his (integer programming) column generation code spending over 90% of the total CPU time with the pricing problems. Although not consistently, this trend was also observed in part by other authors. Reason enough for us to investigate this bottleneck procedure in more detail. It should be stated clearly, however, that it is not necessarily the subproblem that imposes the largest computational burden to a column generation code. With the restricted master program growing bigger it may be a formidable task even for state-of-the-art solvers to cope with the resulting linear programs, c.f. Subsection 2.3.4.

Single Subproblem

Notwithstanding the large cardinality of Q (and E) it is reasonable not to assume that columns represent unstructured or even random data. This is certainly true for the various application areas one finds in the literature, where columns encode cutting patterns, crew pairings, or vehicle tours, to name only a few. Otherwise, the task of gathering data for a truly large linear program would be almost hopeless. For the methodology we have so far discussed it is most natural to think of columns as being drawn from a convex polyhedral set, possible intersected with the integers.[8] In fact, the subproblem (PP) in classical DANTZIG-WOLFE decomposition amounts to solving

$$\begin{array}{lrcl} z^\star := \min & (\boldsymbol{c}^\mathsf{T} - \boldsymbol{u}^\mathsf{T}A)\boldsymbol{x} & & \\ \text{subject to} & D\boldsymbol{x} & \leq & \boldsymbol{d} \\ & \boldsymbol{x} & \geq & 0\ , \end{array} \tag{2.9}$$

which is a linear program over the polyhedron defined by the reformulated subsystem S that can be solved for example by the simplex method. For the reason of a simple statement we chose $K = 1$. Since we assumed non-emptiness of S and keeping in mind that the dual variable v corresponds to the convexity constraint of the master program, three cases are of interest, *viz.* (a) $z^\star \geq v$, (b) $z^\star < v$ and finite, and (c) $z^\star$ is unbounded from below. In case (a) it is proved that no column "of the desired form" exists with negative reduced cost coefficient. The other two cases give rise to an admissible column to be adjoined to the (RMP$'$), since we either obtain an extreme point or an extreme ray, respectively, of S. By employment of the transformation introduced in Subsection 2.1.1 on page 28, the new column is of the form

$$\text{(b)} \begin{pmatrix} \boldsymbol{c}^\mathsf{T}\boldsymbol{x}^\star \\ A\boldsymbol{x}^\star \\ 1 \end{pmatrix}, \qquad \text{or} \qquad \text{(c)} \begin{pmatrix} \boldsymbol{c}^\mathsf{T}\boldsymbol{r}^\star \\ A\boldsymbol{r}^\star \\ 0 \end{pmatrix}, \tag{2.10}$$

respectively, depending on whether the optimal solution to (2.9) is an extreme point $\boldsymbol{x}^\star$ or a *homogeneous solution* to (2.9), i.e., $D\boldsymbol{r}^\star \leq 0$, $\boldsymbol{r}^\star \geq 0$. Recall, that the latter is readily available from the simplex method when the optimal objective function value is unbounded. One should be aware of the fact that neither the extreme points nor the extreme rays provided by the subproblem need to be extreme points or rays, respectively, of the master program in which they are put together by convex/non-negative combination.

Block Diagonal Structure

In principle, the formerly treated case of a block diagonal matrix D, i.e., $K > 1$, can be handled in a precisely analogous way. Except that, in general, it is much more efficient to exploit the structure and split up the pricing problem in K independent—and smaller—subproblems. We present

[8] Actually, in general, from the polyhedra or discrete points in question we obtain only a well-defined *linear transformation* of an admissible column, c.f. (2.10), which is of course no defect in this beautiful principle.

this possibility with respect to the case of integer programming decomposition as introduced in Subsection 2.1.2.

Denoting by $\boldsymbol{c}^k$, A^k, and $\boldsymbol{x}^k$ the respective segment of $\boldsymbol{c}$, A, and $\boldsymbol{x}$ in accordance to the number n_k of columns of D^k, the k^{th} subproblem, $k = 1, \ldots, K$, reads

$$\begin{array}{lrcl} z_k^\star := \min & (\boldsymbol{c}^{k\mathsf{T}} - \boldsymbol{u}^\mathsf{T} A^k)\boldsymbol{x}^k & & \\ \text{subject to} & D^k\boldsymbol{x}^k & \leq & \boldsymbol{d}^k \\ & \boldsymbol{x}^k & \in & \mathbb{Z}_+^{n_k} \ . \end{array} \qquad (\text{PP}_k)$$

In order to obtain the correct reduced cost coefficients we define δ^k equal to one if the optimum of (PP_k) is finite, and zero in the unbounded case. Then, according to DANTZIG's rule, the *entering column*, if any, is determined by an optimal solution to subproblem $\arg\min_{k\in K}(z_k^\star - \delta^k v_k)$, which gives the global minimum among all reduced cost coefficients. Since only *one* subproblem succeeds in delivering a column this way the computation time spend in the other subproblems appears to have been wasted. We will deal with this issue shortly. It is important to note, that in this case each (PP_k) is again an integer program and, in contrast to the subproblems constructed via the decomposition principle in linear programming, it may be difficult to solve. Thus, the efficiency of the integer programming decomposition approach hinges on the property that repeatedly solving the subproblems be *easier* than solving the compact original integer program at once. It goes without saying, that *easier* does not necessarily refer to its meaning in computational complexity theory. GAMACHE, SOUMIS, MARQUIS & DESROSIERS (1999) have up to 300 $\mathcal{NP}$-complete subproblems, and propose partial column generation in analogy to partial pricing, also in order to avoid the generation of many similar columns.

Polyhedral Combinatorics

The happenstance that we have a polyhedral description (here to be intersected with the integers) of the set of feasible columns at hand allows to apply the powerful machinery of *polyhedral combinatorics*, as was hinted in Figure 2.3 on page 30. When the polyhedron describing feasible columns of the (restricted) master program is a well studied object, e.g., a knapsack polytope, the relevant literature can be exploited to strengthen the linear relaxation of the pricing integer programs. For instance, VANDERBECK (1994) is in this situation and implements cutting planes for various applications, c.f. Table 2.1. He admits, however, that the straight forward use of branch-and-bound to solve the pricing problems results in a faster algorithm, possibly due to the small problem sizes or insufficient strength of the cuts. Even pooling and regular removal of cuts from the formulation only led to marginal reductions in computation time. In contrast, JOHNSON, MEHROTRA & NEMHAUSER (1993) more thoroughly investigate their subsystem polytope and report on encouraging results for adding strong valid inequalities to the mixed integer formulation of the pricing problem before using branch-and-bound. Even more efficient it may be to use a *direct*, e.g., a combinatorial method to solve the subproblem, when an appropriate algorithm is available. Integer multi-commodity flow problems may be solved this way, where the pricing problem is a shortest path problem (*see* AHUJA, MAGNANTI & ORLIN 1993). VAL ÉRIO DE CARVALHO (1999) proposes a clever network flow formulation of the cutting stock problem; the

subproblem amounts to solving a minimum cost flow problem. The polynomial solubility cannot, of course, be taken for granted. In extensive formulations of various vehicle routing problems, the pricing problem is an $\mathcal{NP}$-complete constrained shortest path problem, c.f. DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995).

2.4.1 Assessing Column Quality

Since the rôle of the pricing subproblem is to provide a column that prices out profitably or to prove that none such exists, it is a good point to note that *any* column with negative reduced cost, if any, contributes to this aim. In particular, in order to keep the iteration going there is no need to solve (PP) exactly, as is discussed in Section 5.2. With respect to the ability to choose a different pricing rule it is not even mandatory to state the pricing problem precisely the way we did. The next two subsections deal with this additional degree of freedom.

Besides the above elementary purpose of a profitable column it is useful to consider its quality with respect to the overall performance of the column generation approach. Anticipating our investigations in Section 2.6, the final objective quite often is to determine an *integer* solution to the master program, using column generation to provide a lower bound attesting the quality of this solution. The latter is of course no issue in the classical DANTZIG-WOLFE case, but there still, solving the linear program quickly is desirable. In what follows, we are concerned with a *qualitative* perspective, deferring computational and implementation aspects to Chapter 5.

Dual Point of View

Observe that the dual of the restricted master program is (DMP$'$) with rows omitted, hence a relaxation. An optimal dual solution obtained from (RMP$'$) may still violate constraints of the full dual master program, the identification of which is usually called *separation*. Thus, the primal pricing problem is a separation problem for the dual. What is more, the primal simplex algorithm actually performs as a cutting plane algorithm for the dual. This is why column generation is sometimes related to the seminal work of KELLEY (1961) on minimizing a convex function over a closed convex set by a cutting plane method. To clarify matters consider the example in Figure 2.7, which depicts the dual polyhedron.

Primal feasibility of a basic solution $(\boldsymbol{\lambda}_B, \boldsymbol{\lambda}_N)$ to (RMP$'$), i.e., $A_B\boldsymbol{\lambda}_B = \boldsymbol{b}$, $\boldsymbol{\lambda}_B \geq \mathbf{0}$, and $\boldsymbol{\lambda}_N = \mathbf{0}$ interpreted in the dual means that the gradient $\boldsymbol{b}$ of the dual objective function is a non-negative combination of the gradients of the active dual constraints which relate to the primal basic variables. In other words, we always have dual optimality for the respective dual basic solutions encountered in the course of the primal simplex method—which is a well known correspondence. In Figure 2.7 all dual optimal basic solutions satisfy constraint 1 with equality. The unique basic solution which in addition is dual feasible, thus primal optimal, is given by the intersection of the hyperplanes induced by constraints 1 and 5. The crux is that, although being facet defining for the dual, constraints 2, 3, and 4 give not rise to a dual feasible basic solution constructed by the primal simplex method. Loosely speaking, constraints we add to the dual may give an ex-

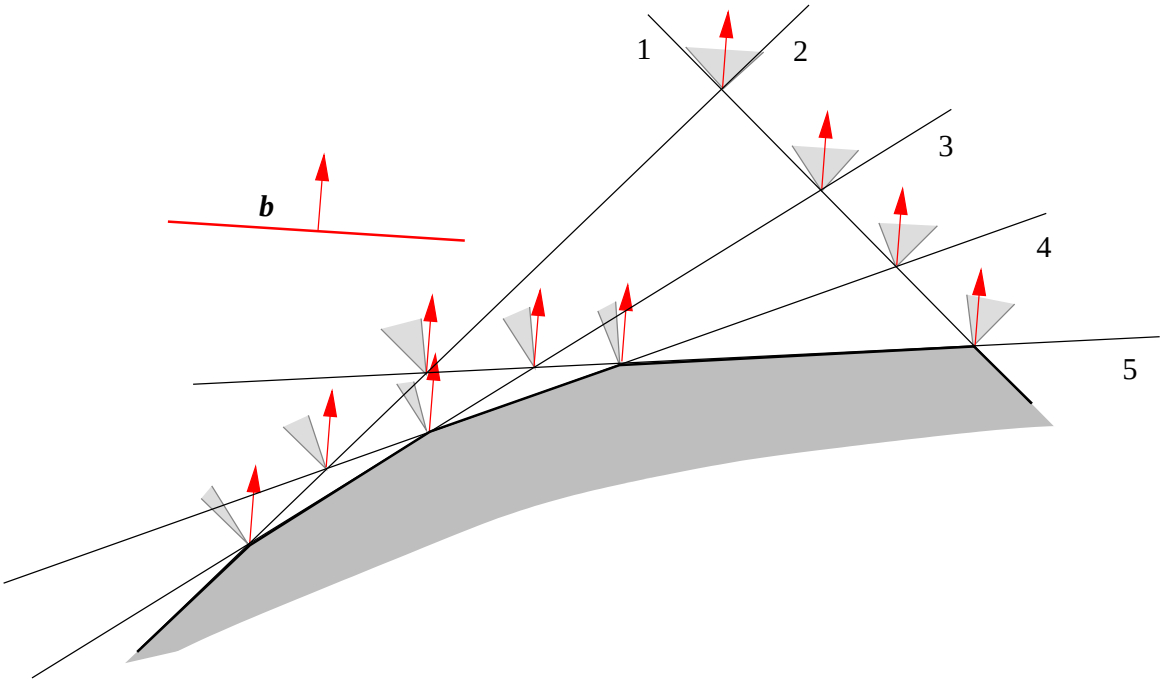

Figure 2.7: Outer approximation of the dual polyhedron using the primal simplex method. By FARKAS' Lemma (*see* SCHRIJVER 1986) an optimal solution is characterized by the gradient of the objective function being a non-negative combination of the gradients of the active constraints.

cellent approximation of the dual polyhedron, but not necessarily in an advantageous region, *viz.* "close" to the dual optimal face. So both, column generation and cutting plane algorithms work with optimal solutions to the respective current polyhedral description, both move towards their respective feasible region, and both suffer from the same problems in doing so. It is sometimes more instructive and revealing to adopt the dual perspective.

Remark. Consequences from the well-known equivalence of separation and optimization, *see* e.g., SCHRIJVER (1986), arise in this context. MINOUX (1987) describes a class of combinatorial problems which can be formulated as set covering/set partitioning problems (of exponential size). The respective linear relaxations are polynomially solvable by column generation under the assumption that the pricing problem is soluble in time polynomial in the number of constraints. In fact, the ellipsoid method is used to solve the dual linear programs of the aforementioned relaxations. Conversely, JOHNSON, MEHROTRA & NEMHAUSER (1993) remark that solving a restricted master program is $\mathcal{NP}$-complete, if solving the pricing problem is.

Dominance of Columns

Particularly *tight* dual constraints are sought by VANDERBECK (1994). He calls a column with reduced cost $\bar{c}$ *dominated*, if there exists another column with reduced cost no larger than $\bar{c}$ for all dual variables ranging within their respective domains. On the other hand, a column with reduced cost $\bar{c}$ is *undominated*, if for all other columns there exists a set of dual variables yielding reduced cost strictly larger than $\bar{c}$. If dominance is detected after the solution of the pricing problem, the column is replaced by the dominating column in a post-processing phase. For instance, in set covering problems a column $\boldsymbol{a}_R$ corresponding to a set $R \subseteq \mathcal{R}$ is dominated, if adding an element $r \in \mathcal{R} \setminus R$ to R incurs no cost, since $c_R - \boldsymbol{u}^\mathsf{T}\boldsymbol{a}_R \geq c_R - \boldsymbol{u}^\mathsf{T}\boldsymbol{a}_R - u_r = c_{R\cup\{r\}} - \boldsymbol{u}^\mathsf{T}\boldsymbol{a}_{R\cup\{r\}}$ for all $\boldsymbol{u} \in \mathbb{R}^m_+$.

Redundancy of Columns

An even stronger concept was introduced in the thesis of SOL (1994). By analogy with the search for *strong*, i.e., high dimensional cutting planes, desirably *facets* of the dual polyhedron in question, one might be tempted to ask for *strong columns* in the primal. SOL therefore calls a column *redundant* if the corresponding constraint is redundant for the dual problem. We repeat his main results here since we are not aware of them being published elsewhere.

Definition 2.4 (Subcolumn Property)
Let S be the set of feasible subsets of a set covering problem, with associated costs c_s, $s \in S$. The pair $(S, \boldsymbol{c})$ satisfies the subcolumn property if and only if for all $s \in S$ it holds that $s \setminus \{t\} \in S$ and $c_s \geq c_{s\setminus\{t\}} - 1$ for all $t \in s$.

This property says that S is closed with respect to taking subsets, and that this latter action strictly decreases the cost of the respective set. If the cost structure only satisfies $c_s \geq c_{s\setminus\{t\}}$ for all $t \in s$, we can redefine $c_s \leftarrow c_s + |s|$, which merely adds a constant $|S|$ to the optimal objective function value.

Theorem 2.5 (Non-Redundant Columns for Set Covering Problems, SOL 1994)
Given a set covering problem with K resources. Let $\bigcup_{k\in K} S_k$ and $\boldsymbol{c}^\mathsf{T} = (\boldsymbol{c}^1, \ldots, \boldsymbol{c}^K)^\mathsf{T}$ satisfy the subcolumn property.

1. *If all resources are identical, then $\boldsymbol{a}_s$ is non-redundant if and only if $\sum_{r\subset s} c_r\lambda_r > c_s$ for all $\boldsymbol{\lambda} \in \mathbb{R}^{2^{|s|}}_+$ with $\sum_{r\subset s} a_{ir}\lambda_r \geq 1$, $i \in s$, and $\lambda_s = 0$.*
2. *In case of all distinct resources,* all *columns are non-redundant.*

The first part of the theorem states that $\boldsymbol{a}_s$ is redundant, if and only if the elements in s can be covered by a non-negative combination of the collection of (the incidence vectors of) all proper subsets of s, the overall cost of which is no greater than c_s. In case of set partitioning problems with identical resources the generation of redundant column can be avoided using an alternative pricing rule.

Theorem 2.6 (Non-Redundant Columns Using an Alternative Pricing Rule, SOL 1994) *Let $S, \boldsymbol{c}$ satisfy the subcolumn property. Furthermore, let $\boldsymbol{\lambda}$ be a solution to the linear relaxation of (SP) such that $\sum_{r \subset s} a_{ir}\lambda_r = 1$, $i \in s$ for a redundant column $\boldsymbol{a}_s$. Then, $\boldsymbol{a}_s$ cannot be an optimal solution to the pricing problem*

$$\min \left\{ \frac{c_s - \boldsymbol{u}^\mathsf{T}\boldsymbol{a}_s}{|s|} \mid s \in S \right\} \quad .$$

What perspective does this theorem offer? SOL (1994) claims that redundant columns can never be part of an optimal solution to the restricted master problem. Although they actually can in the case of the primal optimal solution being dual degenerate, at least, they are not required. However, nothing is said that for a particular dual solution adding a redundant column will not yield a remarkable progress as to the objective function value. We will see in Subsection 2.4.2, that even in this aspect using Theorem 2.6 is a reasonable choice. A much stronger approach to avoid generating columns which cannot appear in an optimal primal solution is presented next. Nevertheless, it will become clear that this approach can greatly benefit from the information of redundant columns.

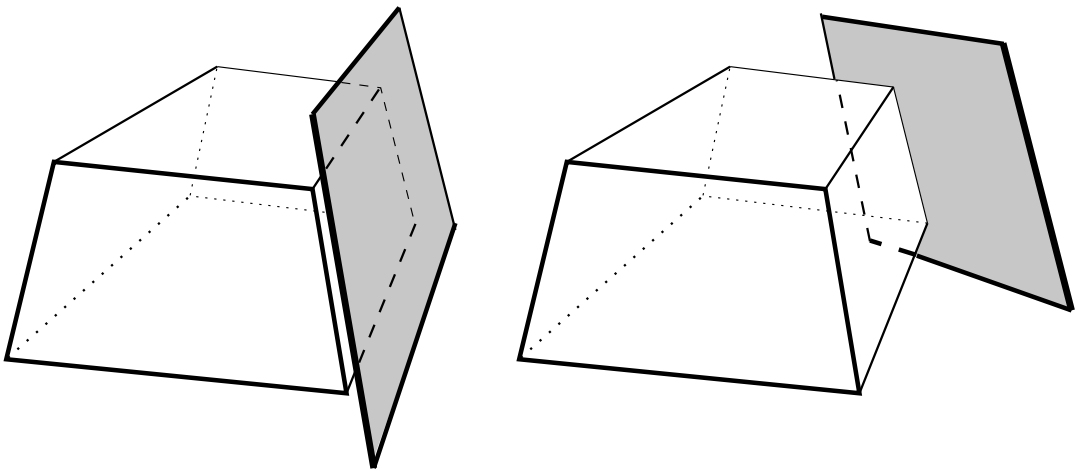

Figure 2.8: Simple examples of the dual geometries of a dominated column (left), the corresponding dual hyperplane of which has the same normal direction as another, tighter, constraint; and a redundant column (right), of which the corresponding dual hyperplane intersects the dual polyhedron in a face of lower dimension, here: an edge. Note, that the redundant column is not dominated in this example.

Dual Cutting Planes

We hitherto investigated adjoining those columns to the restricted master program which emerge from the full master program, i.e., which are given by the problem formulation itself. In order to

obtain a better description of the dual variable space, however, the addition of dual cutting planes comes to mind. To this end, VALÉRIO DE CARVALHO (2000) considers a pair of *extended* primal and dual full master programs of the form

$$\begin{array}{rrcl} \min & \boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda}+\boldsymbol{f}^\mathsf{T}\boldsymbol{y} & & \\ \text{subject to} & A\boldsymbol{\lambda}+F\boldsymbol{y} & = & \boldsymbol{b} \\ & \boldsymbol{\lambda},\boldsymbol{y} & \geq & \boldsymbol{0} \end{array} \qquad \begin{array}{rrcl} \max & \boldsymbol{u}^\mathsf{T}\boldsymbol{b} & & \\ \text{subject to} & \boldsymbol{u}^\mathsf{T}A & \leq & \boldsymbol{c}^\mathsf{T} \\ & \boldsymbol{u}^\mathsf{T}F & \leq & \boldsymbol{f}^\mathsf{T} \end{array} ,$$

where valid inequalities are added to the dual *at initialization time*, i.e., before column generation starts. This corresponds to additional variables $\boldsymbol{y} \geq 0$ in the primal, which are *not* present in the original full master program. From the primal perspective, we therefore obtain a relaxation.

Proposition 2.7 (Recovery of an Optimal Solution, VALÉRIO DE CARVALHO 2000)
Let $p : \{A\boldsymbol{\lambda}+F\boldsymbol{y}=\boldsymbol{b},\boldsymbol{\lambda},\boldsymbol{y}\geq\boldsymbol{0}\} \to \{A\boldsymbol{\lambda}=\boldsymbol{b},\boldsymbol{\lambda}\geq\boldsymbol{0}\}$, $p(\boldsymbol{\lambda},\boldsymbol{y}) \mapsto \boldsymbol{\lambda}$ be a cost preserving mapping, i.e., $\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda}+\boldsymbol{f}^\mathsf{T}\boldsymbol{y}=\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda}$. For any optimal solution $(\boldsymbol{\lambda}^\star,\boldsymbol{y}^\star)$ to the extended master program, an optimal solution to the original master program is given by $p(\boldsymbol{\lambda}^\star,\boldsymbol{y}^\star)$.

The assumption is intuitive, and the result is easily proven. By strong duality it immediately follows from Proposition 2.7 that at least one optimal solution to the original dual master program remains feasible (and optimal) for the extended dual master program. To clarify matters, the question asked here is not for good quality columns to be added during the column generation process, but for reducing the set of admissible columns altogether. In other words, a *good* restriction of the dual polyhedron is sought, ideally (which is illusive) to the optimal face. This simple idea is of a praiseworthy beauty, since it relates three major topics of this chapter. It aims at providing *better* dual solutions, thus speeding up the generation of each column, and enabling a quicker construction of an optimal solution to the master program. It should be noted that the size of a (primal) basis is not affected.

Devising dual cutting planes for a given master program requires and exploits specific problem knowledge, of course. Consider the one-dimensional *cutting stock problem*, in the context of which this technique is developed. Given are paper rolls of width W, and m demands b_i, $i = 1,\ldots,m$, for orders of width w_i, $i = 1,\ldots,m$. The goal is to minimize the number of rolls to be cut into orders, such that the demand is satisfied. A standard (extensive) integer programming formulation is

$$\min\{\boldsymbol{1}^\mathsf{T}\boldsymbol{\lambda} \mid A\boldsymbol{\lambda} \geq \boldsymbol{b},\ \boldsymbol{\lambda} \in \mathbb{Z}_+^{|Q|}\} \ , \tag{2.11}$$

where A columnwise encodes the set Q of feasible *cutting patterns*, i.e., $a_{ij} \in \mathbb{Z}_+$ denotes how often order i is obtained when cutting a roll according to $j \in Q$. In other words, $\sum_{i=1}^{m} a_{ij}w_i \leq W$, must hold for every $j \in Q$, and λ_j determines how often cutting pattern $j \in Q$ is used. The linear relaxation of (2.11) is classically solved via column generation.

Recall, that we strive to prevent columns from being generated, that need not, or better cannot be part on an optimal basis. With respect to the cutting stock problem, it is not sensible to exceed the demand for a certain order length w_{s^*}, and cut this order again (at no extra cost) only to fulfill the demand of a set S of smaller orders, i.e., $\sum_{s\in S} w_s \leq w_{s^*}$, especially when this inequality is strict. This observation motivates the following.

Proposition 2.8 (Dual Cuts for the Cutting Stock Problem, VALÉRIO DE CARVALHO 2000) *Consider the extension of the cutting stock problem formulation (2.11) with the following dual cutting planes added:*

$$-u_{s^*} + \sum_{s \in S} u_s \leq 0, \quad \text{for all } s^*, S \text{ such that } \sum_{s \in S} w_s \leq w_{s^*} \quad . \tag{2.12}$$

1. *For any feasible primal solution $(\boldsymbol{\lambda}, \boldsymbol{y})$ to the linear relaxation of the extended formulation, there exists a feasible solution to (2.11) with the same cost, in particular when $\boldsymbol{y} \neq \mathbf{0}$.*

2. *The dual cutting planes (2.12) are valid inequalities for the optimal face of the polyhedron associated to the linear relaxation of the dual of (2.11).*

The constructive proof of the proposition's first part gives a procedure for actually recovering a feasible solution to (2.11). In particular, the objective function value of the recovered optimal solution remains the same. When applying only (the simplest) $O(m)$ dual cutting planes, VALÉRIO DE CARVALHO (2000) reports on a considerable speedup for solving the linear relaxation of (2.11). The author observes an increased number of pivot steps needed to re-optimize the restricted master program when a column was added. The impact of the increased problem symmetry, c.f. Section 3.1.1, to finding integer solutions is not evaluated. It should be mentioned, that HOLMBERG & JÖRNSTEN (1995) proposed to use dual cutting planes in DANTZIG-WOLFE decomposition for non-linear programming problems. Transferring their idea to the linear situation, a cut is derived from the original solution $\hat{\boldsymbol{x}} = \sum_{q \in Q'} \boldsymbol{p}_q \lambda_q$ by adding the column $(c(\hat{\boldsymbol{x}}),\ A\hat{\boldsymbol{x}} - \boldsymbol{b})^{\mathsf{T}}$ to the restricted master program.

Remark. Related but less sophisticated is the following. Set partitioning type master programs, c.f. Section 2.1.3, can be converted to set covering type, preserving the optimal objective function value, when $c(\boldsymbol{x}_{R_1}) \leq c(\boldsymbol{x}_{R_2})$ for $R_1 \subseteq R_2$. This, in effect, constrains the dual space by restricting the dual variables in sign, and possibly produces the aforementioned benefits, c.f. Figure 2.9. *See* also our imposing bounds on the dual variables at initialization time in Subsection 2.3.3.

Remark. Primal degeneracy may cause the simplex method, and thence column generation, to *stall* at a basic solution, consecutively bringing promising *zero activity* columns into the basis. Each of these bases corresponds to an alternative dual solution, and the hope is that a restricted dual space partly restrains primal degeneracy as well. Preliminary experiments confirm this.

Remark. BRUSCO & JACOBS (1998) specify characteristics of redundant columns, the corresponding variables of which cannot be part of an optimal solution of an *explicitly stated* linear program arising in continuous tour scheduling. RYAN & FALKNER (1988) exemplify how excluding certain columns naturally increases the chance of obtaining integer optimum solutions. MINGOZZI, BIANCO & RICCIARDELLI (1997) proceed with similar aim for a multiple depot vehicle scheduling problem. Although the original motivation is different such information could

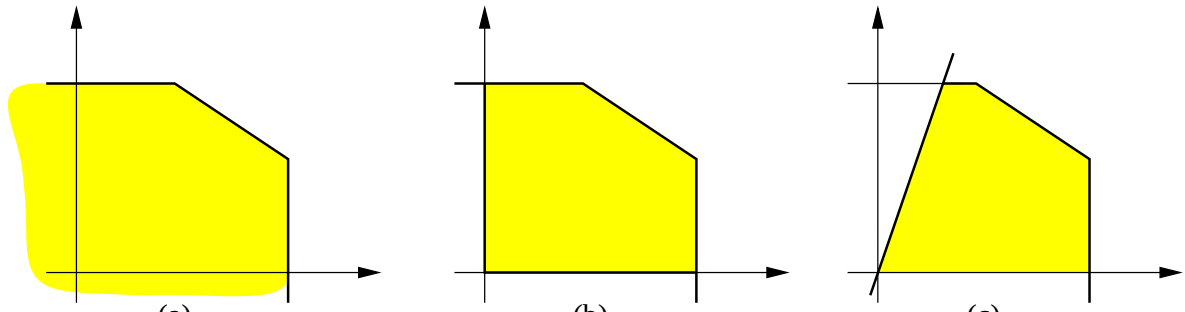

Figure 2.9: Depicted are polyhedra of feasible dual solutions. Part (a) shows the case of an equality constraint restricted master program. When these equalities are eligible for a relaxation to inequalities, dual variables become restricted in sign, here corresponding to the application of the dual cuts $-u_i \leq 0$, c.f. Part (b). In Part (c), finally, adding more elaborate dual cutting planes further restricts the dual space; however, the dual optimal face (at least one vertex of which) remains intact.

also be used in our context. In particular, it would be interesting to investigate if the argumentation, which stems from common sense, leads to a stimulus for dual cutting planes as in the case of the cutting stock problem.

Geometric Dual Interpretation and the Hull Approach

In the context of resource constrained shortest path problems, c.f. Subsection 4.2.2, MEHLHORN & ZIEGELMANN (2000) introduce an amazing interpretation of a very particular dual variable space. Their point of view is motivated by geometric duality. Consider the following simple binary master program with two constraints

$$\min\{\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda} \mid \mathbf{1}^\mathsf{T}\boldsymbol{\lambda} = 1,\ \boldsymbol{r}^\mathsf{T}\boldsymbol{\lambda} \leq \rho,\ \boldsymbol{\lambda} \in \{0,1\}^{|Q|}\} \quad , \tag{2.13}$$

where a minimum cost choice of an element from the set Q is sought such that the corresponding component in $\boldsymbol{r}$ does not exceed ρ. Associated with the constraints of the linear programming relaxation of (2.13) is a pair of dual variables $(u,v) \in \mathbb{R} \times \mathbb{R}_-$. The parameters of each column can be represented in the r-c plane, and each dual constraint $r_j v + u \leq c_j$, $j \in Q$ is interpreted as a *point* (r_j, c_j) in that plane. This gives rise to interpret explicit values for u and v as the *line* $rv + u = c$ with non-positive slope v and c-abscissa u. In order to describe a dual feasible solution, all points (r_j, c_j), $j \in Q$ must lie above or on the associated line. The dual objective is to maximize $u + \rho v$. Consequently, an optimal dual solution is the line spanned by the segment of the lower envelope of $\{(r_j, c_j)\}_{j \in Q}$ which intersects the line $r = \rho$, c.f. Figure 2.10 (a).

Figure 2.10 (b) depicts a strategy to construct the line in question. The method, *de facto*, is column generation. Provided, an initial line, i.e., a dual solution, exists which is feasible for (r_j, c_j), $j \in Q' \subseteq Q$, where possibly $Q' = \emptyset$. The current line is moved in normal direction until an extreme point $(r_{j'}, c_{j'})$ of the lower envelope is hit or it can be proved that already all (r_j, c_j),

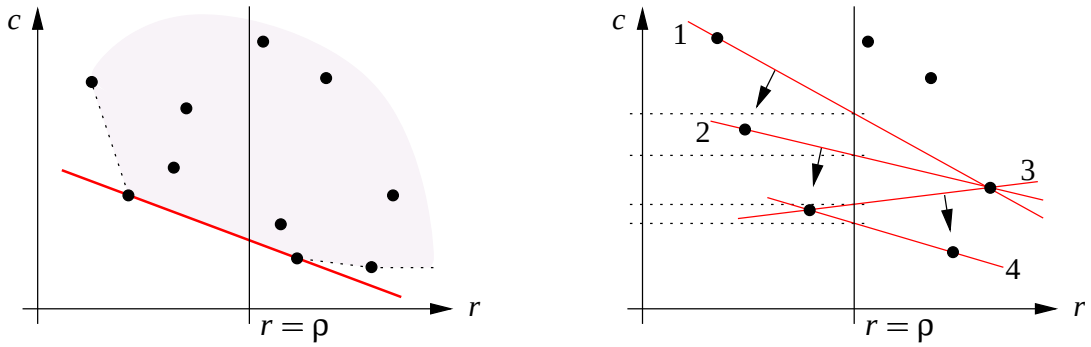

Figure 2.10: Geometric dual interpretation of the dual variable space. The segment of the lower hull depicted in Part (a) intersects $r = \rho$, i.e., gives maximal dual objective function value. Part (b) sketches the proceeding of finding this segment by iterative construction of lines, i.e., dual solutions, in the mentioned order. Note, that the evaluation of these lines at $r = \rho$ gives a (strictly) decreasing sequence of c-values, representing the primal objective function value.

$j \in Q$ lie on or above the current line. The newly found extreme point, together with one of the two points describing the current line, gives a new line, feasible for $Q' \cup \{j'\}$, and the process is iterated. In the case of MEHLHORN & ZIEGELMANN (2000) the movement of a line simply amounts to a polynomial time shortest path computation. Furthermore, the authors prove that a polynomial number of iterations suffice to reach the optimum.

2.4.2 Alternative Pricing Rules

As indicated in Theorem 2.6 deviating from the use of the classical DANTZIG rule may present an advantage from a qualitative point of view. As we remarked, the results from the preceding subsection are related to the *strength* of the dual polyhedral description and do not point out that the actual choice of these columns to be iteratively incorporated in the restricted master program will result in approaching an optimal solution fast—or even, in a sense, *fastest*. This latter issue is approached in the sequel. Again, in our search for a *good guide* for column selection, there is a primal and a dual perspective. Let us start with the former.

Steepest-Edge

Denote by $\bar{A}$ the full constraint matrix of the master program, including convexity constraint(s). For the sake of simplicity, let variables $\lambda_1, \ldots, \lambda_m$ be basic, and variables $\lambda_{m+1}, \ldots, \lambda_{|Q|}$ be non-basic. From the familiar presentation $\bar{A}_B\boldsymbol{\lambda}_B + \bar{A}_N\boldsymbol{\lambda}_N = \bar{\boldsymbol{b}}$ we readily obtain

$$\begin{array}{rcrl} \boldsymbol{\lambda}_B & = & \bar{A}_B^{-1}\boldsymbol{b} - \bar{A}_B^{-1}\bar{A}_N\boldsymbol{\lambda}_N & (\geq \ \mathbf{0}) \\ \boldsymbol{\lambda}_N & = & I\boldsymbol{\lambda}_N & (= \ \mathbf{0}) \end{array}$$

for any feasible basic solution. The *edge directions*

$$\boldsymbol{\eta}_j = \begin{bmatrix} -\bar{A}_B^{-1}\bar{A}_N \\ I \end{bmatrix} \boldsymbol{e}_j, \quad j = m+1,\ldots,|Q|$$

lend their name from the fact that increasing a non-basic variable corresponds to moving from the extreme point associated with the current basis along an edge of the polyhedron of feasible solutions. Note, that in case of primal degeneracy not all edge directions need to be feasible, thus zero step lengths are allowed. With $\boldsymbol{c}^\mathsf{T} = (\boldsymbol{c}_B, \boldsymbol{c}_N)^\mathsf{T}$, the usual reduced cost coefficients

$$\boldsymbol{c}^\mathsf{T}\boldsymbol{\eta}_j, \quad j = m+1,\ldots,|Q|$$

give the change in the objective function per unit change of the respective non-basic variable λ_j. Picking a variable according to DANTZIG's rule means greedily choosing a direction of steepest gradient. The delusive about this measure is that a unit change along the λ_j-axis does not correspond to a unit change along the edge direction $\boldsymbol{\eta}_j$. In other words, the selected edge may locally seem profitable but actually is *short*—and this is not detected. Nevertheless, this can be amended simply by taking into account the direction of an edge, namely considering

$$\frac{\boldsymbol{c}^\mathsf{T}\boldsymbol{\eta}_j}{\|\,\boldsymbol{\eta}_j\,\|_2}, \quad j = m+1,\ldots,|Q| \tag{2.14}$$

as profit valuation of a non-basic variable λ_j. Choosing a variable $\boldsymbol{\lambda}_{j^\star}$ that minimizes (2.14) as entering variable is called *steepest-edge pricing*, because along the edge $\boldsymbol{\eta}_{j^\star}$, the objective function decreases most rapidly with respect to distance in the space of *all* the variables. In fact, (2.14) is the *directional directive* of $\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda}$ along $\boldsymbol{\eta}_j$. While this natural idea dates back at least to the 1960s, a straight forward implementation is prohibitive from a computational standpoint. HARRIS (1973) was the first to present a practicable approximation to (2.14) which was incorporated in her linear programming code named `Devex`. It was not until the paper by GOLDFARB & REID (1977) that the first practicable exact steepest-edge algorithm was described. Their idea is to update the norms of the edge directions via recurrence formulae, thus drastically decreasing the computational efforts spent per iteration. More recent computational studies of FORREST & GOLDFARB (1992) "demonstrate unambiguously the superiority of steepest-edge pivot selection criteria to other pivot selection criteria in the simplex method." SOL (1994) reports that this pricing rule performs particularly well for set partitioning problems.

Deepest-Cut

The intuition of quickly[9] approaching a primal optimum carries over to the dual concept of rapidly reaching dual feasibility by activating appropriate dual constraints. The dual pendant to steepest-edge is called *deepest-cut*, the rationale of which is to activate a dual constraint that

[9] Of course there is no guarantee that steepest-edge reaches an optimum in less time. However, the experiments conducted by FORREST & GOLDFARB (1992) indicate a better performance on the average. Moreover, in general the `Devex` approximation is used for faster computations (BIXBY 2000*b*).

cuts away as much of the dual space as possible. It is important to distinguish this from the ideas concerning a strong dual formulation considered in the preceding subsection. Here, in analogy to the above, we are interested in a locally best decision of what entering variable to select with respect to the progress made by the simplex algorithm. VANDERBECK (1994) also discusses this column selection criterion. Recall in this context also our comparison of central prices and extreme point dual solutions in Subsection 2.3.1.

Consider again Figure 2.7 on page 50. Let the current dual solution $(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}$ be given by the intersection of the hyperplanes induced by constraints 1 and 2. Clearly, activating constraint 4 or even 5 should be preferred to activating constraint 3. More precisely, we would like to choose a dual constraint, whose induced hyperplane has maximal distance to $(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}$. Again, denote by $\bar{A}$ the full constraint matrix of the master program. Let $c_q - (\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q < 0$, i.e., column $\bar{\boldsymbol{a}}_q$ corresponds to a violated constraint in the dual. The EUCLIDEAN distance of $(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}$ to the hyperplane given by $c_q = (\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q$ is the absolute amount of violation of the constraint divided by the norm of its normal vector, *viz.*

$$\frac{(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q - c_q}{\| \bar{\boldsymbol{a}}_q \|_2} \quad . \tag{2.15}$$

Some remarks affirming the usefulness of (2.15) are in order here. At first, note, that the pricing rule stated in Theorem 2.6 is a deepest-cut criterion. Thus, under the assumptions of the theorem, this criterion is not only in conformance with our geometric intuition but also gives a cut of provable best quality. Secondly, in case of set partitioning/covering problems, $\bar{\boldsymbol{a}}_q$ is a binary vector and computation of the norm amounts to adding its non zero components. Therefore, this norm is much simpler to determine than the one used in (2.14). At least in this special case, this simplicity contrasts the claim of VANDERBECK (1994) this former measure be computationally expensive.

The deepest-cut criterion admittedly has its drawbacks. The distance given by (2.15) is the distance between $(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}$ and its orthogonal projection $(\hat{\boldsymbol{u}},\hat{\boldsymbol{v}})^\mathsf{T}$ on the hyperplane defined by the violated cut. If $(\hat{\boldsymbol{u}},\hat{\boldsymbol{v}})^\mathsf{T}$ is not dual feasible, however, this measure becomes pointless. Intuition suggests again that it is misleading to judge the dual solution, which is not directly operated upon in the primal method we use. More formally, $-\bar{A}_B^{-1}\bar{\boldsymbol{a}}_q = (\boldsymbol{\eta}_q)_B$ establishes that the steepest-edge and the deepest-cut criterion basically provide the same measure, the former, however makes use of the current primal solution, while the latter merely works with the original data. That is, barring computational efforts, one would opt for steepest-edge. It must be stated, however, that the steepest-edge criterion is inherently based on the simplex method, whereas the deepest-cut rule is more independent from a particular solution method.

Lambda Pricing

In desire for a column selection strategy with this latter property (independence of solution methods) BIXBY, GREGORY, LUSTIG, MARSTEN & SHANNO (1992) introduced a rule they termed *lambda pricing*. It is again of the *normalized reduced cost* type. The authors assume that $c_q \geq 0$,

$q \in Q$. Clearly, the reduced cost $c_q - (\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q$ are non-negative for all $q \in Q$ if and only if

$$\min_{q\in Q}\left\{ \frac{c_q}{(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q} \mid (\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q > 0 \right\} \quad \geq 1 \ . \tag{2.16}$$

At first glance, this is just a reformulation. However, (2.16) takes advantage of structural properties of set partitioning problems, for which it was designed. Intuitively, picking columns with small $c_q/(\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q$ accounts for smaller cost coefficients as well as for more non-zero entries in $\bar{\boldsymbol{a}}_q$. In this particular case, columns corresponding to an optimal solution predominantly had this character. BIXBY, GREGORY, LUSTIG, MARSTEN & SHANNO (1992) report on a reduction of the number of major iterations of their sifting method, c.f. Subsection 2.3.4, when using lambda pricing. DESROSIERS (1999*a*) observed a mild speedup also for standard column generation.

Lagrangian Pricing

With *Lagrangian pricing*, LÖBEL (1997, 1998) proposes a clever way of exploiting the problem information provided by the *original* formulation. The rationale behind this concept is to generate columns based not only on the local reduced cost criterion, but also to take into account a possible interaction with other variables. The general method—omitting several enhancements suggested by LÖBEL—proceeds as follows.

repeat
  Obtain dual (optimal) solution $\boldsymbol{u}^\star$ to (RMP′)
  Obtain primal (optimal) solution $\boldsymbol{x}^\star$ to a LAGRANGIAN relaxation of (CF) using $\boldsymbol{u}^\star$
  Deduce from $\boldsymbol{x}^\star$ columns to be adjoined to (RMP′)
  Re-optimize (RMP′)
until decrease in objective function value $\boldsymbol{c}^\mathsf{T}\boldsymbol{\lambda}$ stalls
Fall back to some *standard* pricing scheme

Already from this presentation it becomes clear that considerable customization is necessary, e.g., deciding which LAGRANGIAN relaxation(s) to use. Furthermore, the deduction of columns, let alone *good* ones, from $\boldsymbol{x}^\star$ can be non-trivial. Nontheless, the charm of using LAGRANGIAN relaxations of the compact formulation rests upon controlling the structure of added columns. LÖBEL reports that the so-called LAGRANGIAN phase performs poorly when close to optimum. Therefore, column generation as usual succeeds this phase. Very large scale linear programs emerging from practical vehicle routing problems are optimally solved using this pricing scheme.

A proposal by SWEENEY & MURPHY (1979) can be regarded as ancestor of LAGRANGIAN pricing for problems with block diagonal matrices with coupling constraints. The latter are relaxed in a LAGRANGIAN fashion, resulting in K independent subproblems. Iteratively, the k_i, $i = 1,\ldots,K$ best solutions to the subproblems give rise to columns to be added to the restricted master program. The authors show that an optimal solution is reached as soon as the cost difference between optimal and k_i^{th} best solution for each subproblem $i = 1,\ldots,K$ is not greater than the duality gap, calculated from the primal solution and the lower bound obtained from LAGRANGIAN relaxation. Ranking the subproblem solutions is the complication of this technique.

Remark. A problem specific, yet simple, rule was proposed by GILMORE & GOMORY (1963) for the cutting stock problem to circumvent an only relatively small increase of the selected non-basic variable λ_j. Their *median method* forms two groups of rows, according to a *large* or a *small* ratio between b_i and a possible a_{ij}, respectively, which is easy e.g., when A is binary. Every other iteration, only rows in the "large ratio" group are considered, heuristically allowing a larger increase of λ_j.

Remark. Many publications on simplex pricing are available, only a few of which have been related to column generation. For instance, SWOVELAND (1974) proposes to generate columns which maximally increase the objective function value. However, the computational usefulness of such comparably aged proposals needs assessment from a modern implementation point of view.

Good Columns Aiming at Integer Master Solutions

All attempts to measure the profitability of entering columns presented so far relate to solving a linear master program. Ultimately, however, in the practical applications we are usually interested in solving the master integrally. An important thing to note is that precluding a subset of columns to be generated may result in integer infeasible restricted master programs. For instance, Theorem 2.6 holds only when no variables are fixed, for precisely this reason (SOL 1994).

HOLM & TIND (1988) come up with penalizing the violation of integrality constraints. The idea is to introduce possibly redundant integral bounds, say zero and one, on the *original* variables. These constraints are expressed in terms of the master program, and the associated dual variables are used to set prices on non-integral solutions. HURKENS, DE JONG & CHEN (1997) introduce a problem specific *penalty function method*, i.e., roughly sketched, they incorporate an integrality violation penalizing non-linear summand in the objective function of the pricing problem. That is, this method aims at the construction of columns with integral coefficients. The summand takes the form $\rho \cdot \sum_{i=1}^{n} (a_{iq} - \lfloor a_{iq} \rfloor)$ with a penalty parameter ρ. Global convergence of the approach is proved, and it is solved via feasible direction methods. It remains to be evaluated if a similar proceeding is feasible for enforcing integral master solutions.

In what concerns the fast solution of the *linear* program, VANDERBECK (1994) takes the view that the simplex method itself seems to be the best mechanism for generating appropriate columns. Adjoining additional columns that are presumably part of an (optimal) integer solution interferes with this mechanism by influencing its "guide," namely the dual variable values. Moreover, the unnecessarily enlarged restricted master program requires longer solution times. VANDERBECK based this assessment on computational results he obtains when initializing the column generation process with (a) an optimal integer solution, and (b) a set of columns likely to be part of an integer optimal solution. VANCE *et al.* (1997) state compatible observations.

For all that, the concept of adjoining *partial solutions* to the restricted master in the course of column generation seems to offer significant advantages as for obtaining integer solutions. DESROCHERS, DESROSIERS & SOLOMON (1992) observe that adding columns to a restricted

set covering master program that represent "almost disjoint sets the efficient discovery of integer solutions is enhanced." What is more, SAVELSBERGH & SOL (1998) point out the reduction in similar columns added when the dual solution is far from optimal. Recently, DESROSIERS (1999*b*) acknowledged good experience with adjoining columns having "the structure of the desired or expected solution."

Concluding Remarks

We have seen that not all admissible columns serve our goals equally well. The natural question for those which do best is not consistently to answer owing to the multiple, sometimes contrary, evaluation criteria. Thus, advising a strategy strongly depends on (experience with) the particular application. In the end, computational efforts may cancel theoretical benefits.

Concluding, we wish to point out some gaps in the theory concerning the subjects of this section, which have not been closed in the literature to the best of the author's knowledge. Common to all presented pricing rules is their sensitivity to the dual variable values, in case the current basic solution admits multiple dual solutions, as discussed in Subsection 2.3.1. This is an explanation for the experience of e.g., NEMHAUSER (1999) that for large set partitioning problems—which are highly primal degenerate—the value of the dual variables give not rise to a meaningful measure for which column to adjoin to the (RMP′). The only concept that offers compensation for this sensitivity so far is to try to remove degeneracy itself, *see* e.g., VALÉRIO DE CARVALHO (2000) and our discussion on dual cutting planes, also Subsection 2.5.4 on stabilization, and DESAULNIERS, DESROSIERS, IOACHIM, SOLOMON, SOUMIS & VILLENEUVE (1998), and the references therein which point to other anti-cycling techniques.

Finally, although primal as well as dual information went into pricing strategies, complementary slackness conditions seem not to having been satisfactorily exploited or applied.

2.5 Aspects of Convergence

Linear programming column generation is known for its poor convergence.[10] While usually a *near optimal* solution is approached considerably fast, only little progress per iteration is made with respect to the objective function value close to the optimum. Also, in relation, it may be quite time consuming to finally prove optimality of a degenerate optimal solution. Figuratively speaking, the solution process exhibits a *long tail*,[11] whence this phenomenon is called the *tailing off effect*.

Starting with a general discussion of computational difficulties encountered in decomposition methods, we use this knowledge in search for an explanation for the tailing off effect, and discuss remedies in Subsections 2.5.2 through 2.5.4, respectively.

[10] It is customary in the column generation literature to speak of *convergence* despite the *finiteness* of the algorithm under the usual non-cycling assumption.

[11] This wording was already used in the seminal work of GILMORE & GOMORY (1963).

2.5.1 Computational Difficulties

At least three types of computational difficulties attributed to column generation are offered in the literature, *see* e.g., KIM & NAZARETH (1991) and NAZARETH (1987). Figure 2.11 serves as an illustration, and we consider a single subproblem only. Let ℓ denote the feasible region, in our example a line, defined by the original constraints $A\boldsymbol{x} \leq \boldsymbol{b}$; let $\boldsymbol{x}^\star$, in terms of the original variables, correspond to the optimal solution obtained from the current (RMP$'$); and let the depicted polyhedron define the feasible region S of the subproblem (assumed bounded here).

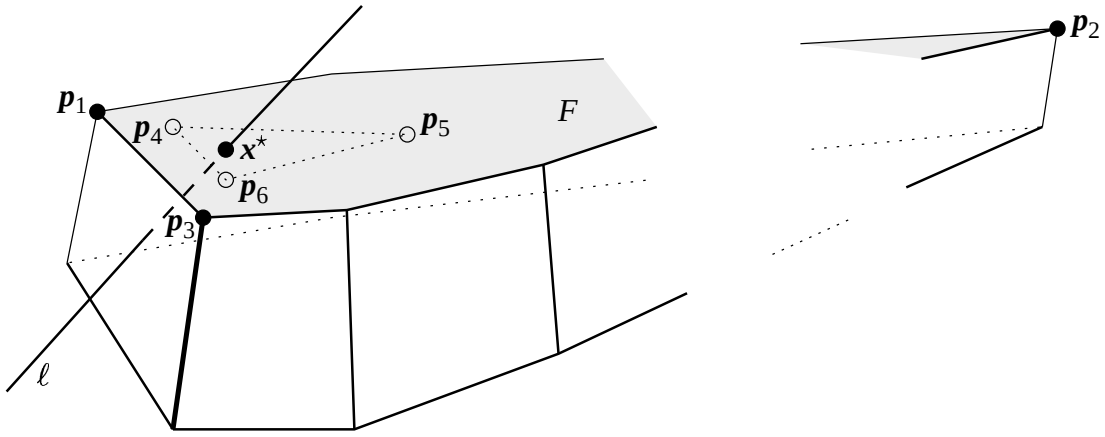

Figure 2.11: Convex combining subproblem to master solutions

Combinatorics

The first difficulty is *combinatorial* in nature. According to (2.1), $\boldsymbol{x}^\star$ is a convex combination of subproblem extreme point solutions, $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, and $\boldsymbol{p}_3$ in Figure 2.11, here contained in a face F of S. KIM & NAZARETH (1991) attach remarkable computational efforts to find a feasible convex combination to the possibly "complex combinatorial structure" of faces of S. This complication may be even more severe in presence of several subproblems. HURKENS, DE JONG & CHEN (1997) introduce a problem specific reformulation of the pricing problem in order to *a priori* restrict attention to a set of known *simple* columns; *see* also BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998) for a similar technique.

Geometry

Second is the already observed *geometric* fact that the optimum $\boldsymbol{x}^\star$ to (CF) is approached possibly through the interior of the corresponding original polyhedron $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b},\ D\boldsymbol{x} \leq \boldsymbol{d},\ \boldsymbol{x} \geq \boldsymbol{0}\}$. Considerable "cleaning up" (NAZARETH 1987) is required to obtain an optimal *basic* solution to (CF), i.e., to exactly satisfy the complementary slackness condition. We develop here a more

formal description of this paraphrase, c.f. Figure 2.12. Denote by $\boldsymbol{w}$ the dual variables associated with subproblem constraints $D\boldsymbol{x} \leq \boldsymbol{d}$ in (CF), and let B_2 be an arbitrary basis to the latter system. Since $\boldsymbol{w}^\mathsf{T} D_{B_2} = (\boldsymbol{c}^\mathsf{T} - \boldsymbol{u}^\mathsf{T} A)_{B_2}$ we have $(\boldsymbol{c}^\mathsf{T} - \boldsymbol{u}^\mathsf{T} A - \boldsymbol{w}^\mathsf{T} D)_j = 0$ for all $j \in B_2$, that is, the reduced cost of any column of $\binom{A}{D}$ with index in B_2 is zero. Moreover, $D_{B_2}\boldsymbol{x}_{B_2} = \boldsymbol{d}$. Now let $\bar{\boldsymbol{x}} = \sum_{q \in B} \boldsymbol{p}_q \lambda_q$ be an original solution constructed from the current basis B of the restricted master program. We define $B_1 = \{j \in \{1,\ldots,n\} \mid x_j > 0\}$ and obtain $A_{B_1}\bar{\boldsymbol{x}}_{B_1} = \boldsymbol{b}$. However, B_1 is not necessarily a basis for the system $A\boldsymbol{x} \leq \boldsymbol{b}$ of linking constraints. In addition, the trouble is that $\boldsymbol{w}^\mathsf{T} D_{B_1 \setminus B_2} = \boldsymbol{0}$ must hold in order to obtain zero reduced cost for $\{x_j\}_{j \in B_1 \setminus B_2}$, i.e., $\bar{\boldsymbol{c}}_{B_1 \setminus B_2} = \boldsymbol{0}$.

We are left with two possible directions to construct a basis for the entire system from B_1 and B_2 (and potentially additional indices, the corresponding variables at level zero), for which complementary slackness conditions are fulfilled; (a) decrease some variables in $\{x_j\}_{j \in B_1 \setminus B_2}$ to zero, allowing for non zero reduced cost for these variables, and (b) decrease some dual variables $\boldsymbol{w}$ to zero, enabling the corresponding constraints in $D\boldsymbol{x} \leq \boldsymbol{d}$ to be fulfilled with strict inequality. Both actions unavoidably influence the respective other variables. Since these changes are applied iteratively, not simultaneously, a certain amount of *tuning* is to be expected. Very small adjustments may be necessary close to optimum.

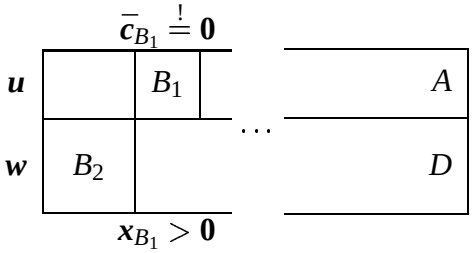

Figure 2.12: Combining B_1 and B_2

In a sense, this problem exposes certain similarities to recovering an optimal basis from a solution provided by an interior point method, *see* e.g., BIXBY & SALTZMAN (1994). Approaches from this area may provide new avenues of research also for column generation algorithms.

Numerics

The third, and in finite precision arithmetic probably the most severe computational difficulty is *numerical instability*. NAZARETH (1984, 1987) gives examples for bad numerical characteristics of the master program in contrast to a well behaving original formulation. Explicitly, the *scaling* of the coefficient matrix may drastically deteriorate, i.e., its entries may differ by several orders of magnitude, possibly resulting in *ill conditioned* basis matrices.[12] This, in turn, influences the accuracy of the calculated primal and dual variable values, and may produce misleading reduced cost coefficients. Also, computing $\boldsymbol{x}^\star$ from ill conditioned subproblem solutions, symbolized by $\boldsymbol{p}_2$ in Figure 2.11, may not even give the true result. In general, this situation is not unlikely to occur since the decomposition itself dictates the removal of constraints from the subproblem by principle. This possibly leads to *distant* extreme points with large coefficients.

As remedy against numerical instability in general KIM & NAZARETH (1991) propose to use interior point solutions to the pricing subproblem, like $\boldsymbol{p}_4$, $\boldsymbol{p}_5$, and $\boldsymbol{p}_6$. This is done with intent

[12] The *condition number* $\| \bar{A}_B \|_2 \cdot \| \bar{A}_B^{-1} \|_2$ of a basis matrix $\bar{A}_B$ serves as indicator as to how strongly variations in the matrix coefficients incur changes of the respective solution of a linear equation system involving $\bar{A}_B$.

to preclude badly scaled columns from being adjoined to the (RMP′). Unfortunately, no proof of convergence of their method is given and the test bed, on which the promising computational results rely, is not representative by far. Note however that, at least in principle, such an approach is suited to *directly* construct basic solutions of (CF)—and thus avoid passing through the interior of the original polyhedron. We refer to LASDON (1970) for further following up this thread. Finally, let us remark that the discretization variant of integer programming decomposition taken by VANDERBECK (1994), c.f. Section 2.1.2, also uses interior point solutions of the subproblem.

Note that these issues are to be judged in a different light when considering combinatorial optimization problems, the linearly relaxed variables of which take values from the interval $[0,1]$ only, and often coefficient matrices and even right hand sides, respectively, are binary.

2.5.2 The Tailing Off Effect

At first glance, the tailing off effect runs counter to intuition, since column generation is nothing else but a particular pricing scheme predominantly used in the simplex method. How come that the simplex method with a standard pricing scheme does not suffer from this effect to the extend usually observed for the former scheme?[13] To put it differently, what is the particularity to column generation making it slow when the solution process approaches the optimal solution? Note, that this does not mean being slower by a constant factor but usually getting progressively slower towards the end. There exists an intuitive assessment of the phenomenon, but a theoretical understanding has only been partly achieved to date; the monographs of LASDON (1970) and NAZARETH (1987) make notable contributions. Let us remark that this problem is not so much an issue when integer programs are to be solved, c.f. Subsection 2.5.3.

The combinatorial and geometric arguments suggest some explanation for long tails. As remarked earlier, the trajectory of solutions $\boldsymbol{x}$ to (CF) as constructed from solutions $\boldsymbol{\lambda}$ to (RMP′) encountered in the course of the algorithm will, in general, pass through the interior of the polyhedron defined by the original formulation (CF). Besides feasibility and monotone improving objective function value, no special requirements are imposed on such interior point solutions, and the combinatorial freedom to select subproblem solutions satisfying these may be vast. However, in order to combine an optimum (basic) solution $\boldsymbol{x}^\star$ to (CF) from subproblem solutions, all these latter have to lie on a hyperplane containing $\boldsymbol{x}^\star$ (we assume boundedness of the subsystem polyhedron here). The pricing problems being independent, coordinated by the master only, this implies a considerable combinatorial difficulty. In this context, LASDON (1970) points to the observation that linear programs "with many columns often have a tremendous number of near optimal solutions prior to optimality."

HO (1984) considers numerical inaccuracy in the computation of subproblem solutions. He argues that numerical errors may lead to a persistent positive duality gap, and improvements—indicated by negative reduced cost coefficients—may actually be only *noise*. Although being an amplification for the tailing off effect, and certainly affecting the finiteness of column generation,

[13] Actually, long *tails* have been observed for the simplex method as well. However, column generation seems to amplify this effect to such an amount that it became characteristic for the technique.

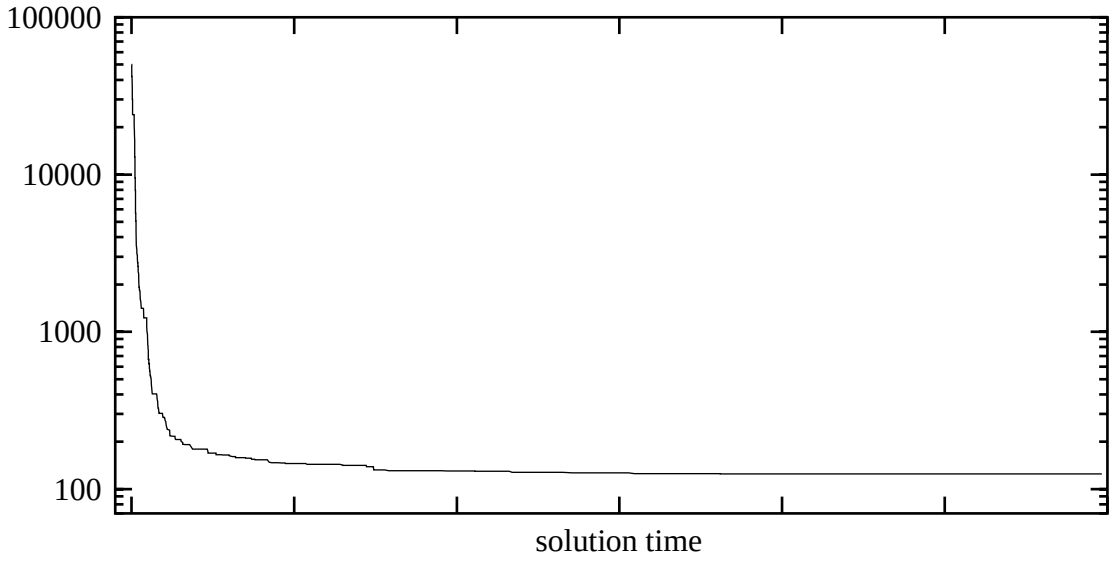

Figure 2.13: The depicted development of the objective function value is typical for the tailing off effect. Note the logarithmic scale.

numerical instability cannot be solely responsible for slow convergence. The latter shows up also when the columns of the restricted master program are computed exactly, as is the case when using e.g., combinatorial algorithms.

HARRIS (1973) evaluates the length of tails for the simplex method, and observes that the `Devex` rule clearly outperforms DANTZIG's rule. It would be interesting to compare the progression in terms of the objective function value made by the simplex method with a standard pricing scheme and its corresponding column generation sibling, respectively, when applied to the same (restricted) master program. How differ pivot steps and per iteration times? Possibly such an investigation could feed back to pricing schemes in general.

ADLER & ÜLKÜCÜ (1973) prove that the diameter of the polyhedron associated with (MP) is not smaller than that of the polyhedron corresponding to (CF). They argue that this is a partial explanation for the tailing off effect, considering that the diameter of a polyhedron is the maximal number of steps an *ideal* vertex following algorithm will take.

Remark. BOUCHER & SMEERS (1986) explicitly conclude that numerical problems and slow convergence do not materialize in their particular column generation application. This, however, contrasts the authors' observation that about one third of the iterations is sufficient to "arrive quite close to optimality" while the rest is "necessary in order to achieve convergence of the dual variables." Thus, in our opinion, their conclusion is questionable.

Oscillation of Dual Variables

In the course of our discussion we repeatedly stressed the importance of the dual solution to the overall performance of column generation. In computational testing, it has been observed that the dual variable values do not *smoothly* converge to their respective optima, but vehemently oscillate, seemingly following no regular pattern. This behavior is regarded as major efficiency issue, and its absence is seen as a (possibly *the*) desirable property; *see* Figure 2.14 for an illustration. Fluctuating dual variables can be connected to the coordination between master and subproblem level we described earlier. In fact, the design of dual cutting planes was motivated by the wish for controlling dual variable convergence. We will come back to this point in Subsection 2.5.4.

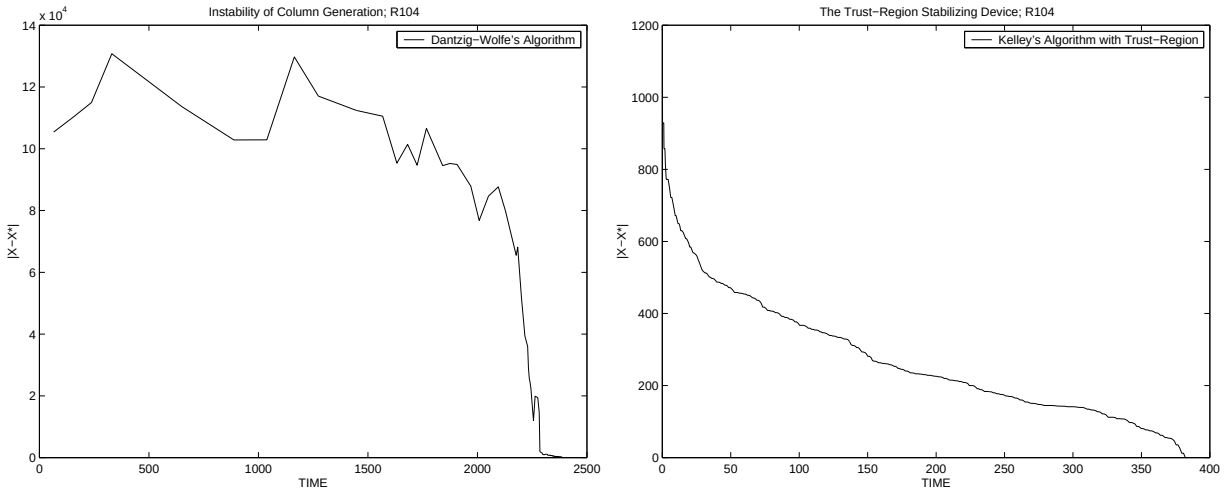

Figure 2.14: Development of $\| \boldsymbol{u}^\star - \boldsymbol{u} \|_2$ plotted against time for two different solution methods applied to the same instance. Observe the different scales. The right picture displays a very stable convergence of the dual variables using the trust region stabilization device, *see* Subsection 2.5.4. The left picture displays the typical dual variable behavior in standard column generation. These plots are kindly provided by BRIAN KALLEHAUGE (2000).

2.5.3 Lower Bounds and Early Termination

Although an explanation of the tailing off effect seems hard to capture rigorously, various amendments have been proposed, especially under the premise of aiming at integer solutions. The very simplest idea to get rid of the long tail is to halt the algorithm *before* tailing off occurs! More precisely, LP column generation can be terminated as soon as a desired solution *quality* can be assured. In this subsection we will investigate bounds that allow such *early termination*.

The effectiveness of requiring a certain relative decrease of the objective function value over a predefined number of iterations—originally proposed by GILMORE & GOMORY (1963)—can be undermined by the intermediate occurrence of stalling. Therefore, besides the absence of a

guarantee, one cannot even *expect* to be reasonably close to the optimum, when the algorithm's progress is temporarily poor. In linear programming the quality of the objective function value can be judged by monitoring the *duality gap*, which of course closes at LP optimality. Alas, in column generation the primal program is restricted and the dual relaxed, hence we have

$$z_{\mathrm{RMP}'} \geq z^\star_{\mathrm{MP}'} = z^\star_{\mathrm{DMP}'} \quad \text{but} \quad z^\star_{\mathrm{DMP}'} \leq z_{\mathrm{DRMP}'} \ .$$

Since the respective interval $[z_{\mathrm{DRMP}'}, z_{\mathrm{RMP}'}]$ of current objective function values needs not contain the sought optimum $z^\star_{\mathrm{MP}'}$, we cannot make use of the aforementioned standard criterion. Instead, a lower bound on $z^\star_{\mathrm{MP}'}$ can be computed in each iteration as follows.

Lemma 2.9 (Lower Bound on $z^\star_{\mathrm{MP}'}$, LASDON 1970)
Let $z_{RMP'}$ denote the current objective function value of a restricted master program with K resources, and let $(\boldsymbol{u},\boldsymbol{v})$ be a corresponding dual solution. Assume the feasible region S_k of the k^{th} subproblem be bounded. Then it holds that

$$z^\star_{MP'} \quad \geq \quad LB_L := z_{RMP'} + \sum_{k=1}^{K} \min\left\{ (\boldsymbol{c}^{k\mathsf{T}} - \boldsymbol{u}^\mathsf{T} A^k)\boldsymbol{x}^k - v_k \mid \boldsymbol{x}^k \in S_k \right\} \quad = \quad z_{RMP'} + \sum_{k=1}^{K} z^\star_k \ .$$

The result can easily be extended to the case of unbounded S_k. This bound and variants have been widely used in the literature, e.g., by SOL (1994) in the context of the *m*-PDPTW. Since $z_{\mathrm{RMP}'} \geq z^\star_{\mathrm{MP}'}$, and the minimal reduced cost coefficients vanish when the optimum for (RMP′) is reached the bound LB_L is tight. Note the resemblance to the LAGRANGIAN dual function (2.4). In fact, $\mathcal{L}(\boldsymbol{u})$ provides the same lower bound as Lemma 2.9 (*see* e.g., LASDON 1970). However, there is a drawback to this result. When e.g., a partial or heuristic pricing is performed, the minimal reduced costs $z^\star_k$, $k \in K$ need not be available, neither so the presented lower bound. The following lemma provides alleviation with this respect at the expense of a slightly increased computational effort.

Lemma 2.10 (Lower Bound on $z^\star_{\mathrm{MP}'}$ when $\boldsymbol{c} \geq \mathbf{0}$, FARLEY 1990)
Given $\boldsymbol{c} \geq \mathbf{0}$, let $(\boldsymbol{u},\boldsymbol{v})$ be an optimal dual solution for (RMP′), and let $z^\star_{\mathrm{RMP}'}$ be the corresponding primal objective function value. Define $\mu_q := (\boldsymbol{u},\boldsymbol{v})^\mathsf{T}\bar{\boldsymbol{a}}_q$, and let $\tilde{q} = \arg\min_{q\in Q'}\{c_q/\mu_q \mid \mu_q > 0\}$. Then a lower bound on $z^\star_{\mathrm{MP}'}$ is given by

$$LB_F := z^\star_{\mathrm{RMP}'} \cdot c_{\tilde{q}}/\mu_{\tilde{q}} \ .$$

Note, that no lower bound is needed in case $\mu_q \leq 0$ for all $q \in Q'$, since this is tantamount to $(\boldsymbol{u},\boldsymbol{v})$ being feasible for (DMP′), hence by strong duality the current $\boldsymbol{\lambda}$ is optimal for (MP′). Moreover, $c_{\tilde{q}}/\mu_{\tilde{q}}$ approaches one from below, finally reaching it; thus the FARLEY bound is tight as well. VANCE, BARNHART, JOHNSON & NEMHAUSER (1994) and VANCE (1998) tailor this result to the cutting stock problem. Also, observe the strong resemblance to the lambda pricing rule (2.16).

Both, LB_L and LB_F are computationally cheap, the former even being almost readily available in every iteration (when exact pricing is performed). Also, both bounds increase—but not monotonically. Since LB_L and LB_F usually differ, monitoring both at the same time is worthwhile, *see* HURKENS, DE JONG & CHEN (1997).

Lower Bounds for Integer Programs

From an integer programming point of view, the only purpose of solving the linear relaxation of the restricted master program is to deliver a lower bound on the integer optimal objective function value. When $c_q \in \mathbb{Z}$, $q \in Q$, which for rational data can always be achieved by an appropriate *scaling*, column generation can be terminated as soon as $\lceil z^\star_{\mathrm{MP}'} \rceil = \lceil z^\star_{\mathrm{RMP}'} \rceil$. Again, $z^\star_{\mathrm{MP}'}$ is not available and $\lceil LB \rceil = \lceil z^\star_{\mathrm{RMP}'} \rceil$ must serve as a practicable criterion, with LB e.g., one of the lower bounds presented above. Furthermore, when column generation is employed within a branch-and-bound framework—this will be done in the next section—and $\bar{z}$ is the incumbent integral objective function value, the current node can be pruned as soon as $\lceil LB \rceil \geq \bar{z}$.

VANDERBECK (1994) and VANDERBECK & WOLSEY (1996) generalize LB_L to the more general case when convexity constraints are replaced by *cardinality constraints*, i.e., lower and upper integer bounds on the sum of all variables. The resulting bound is similar in structure to the above, and is obtained by dualizing all but the convexity constraints in a LAGRANGIAN way. It is slightly stronger than LASDON's. Attesting the quality of the lower bound obtained directly from the restricted master of a set partitioning formulation for vehicle routing problems, BRAMEL & SIMCHI-LEVI (1997) show that the relative integrality gap asymptotically closes when the number of customers increases.

Assuming a single resource in Lemma 2.9, we can compute an *a priori* upper bound on the optimal objective function value of the pricing problem, or equivalently, on the minimal reduced cost coefficient. This is especially useful when the subproblem is indeed an integer program, since an initial *upper cutoff* can be applied to it, potentially helping in accelerating its solution. Column generation terminates as soon as the pricing problem is detected to be infeasible, *see* VANDERBECK (1994) for further details.

Improving the Lower Bound by Valid Inequalities

A standard procedure to improve the lower bound provided by the linear relaxation of an integer program is to add valid inequalities to the formulation. If the cost structure is such that to primarily minimize the number of used resources, a particularly simple one proposed e.g., by DUMAS, DESROSIERS & SOUMIS (1991) is the following.

$$\sum_{k \in K} \sum_{q \in Q_k} \lambda_q^k \geq \left\lceil \sum_{k \in K} \sum_{q \in Q_k} \lambda_q^{k\star} \right\rceil \tag{2.17}$$

It is an advantage of (2.17) that the added inequality does not complicate the pricing problem, since the additional dual variable appears as an additive constant in its objective function value. We will see in Subsection 2.6.1 that this is an utmost desirable property.

Remark. Non-negativity of the reduced cost coefficients may be an inappropriate *stopping criterion* for the column generation algorithm due to round off errors, *see* e.g., MINOUX (1986).

The lower bounds discussed in this subsection supply an alternative to stop when a prescribed optimality tolerance is reached.

2.5.4 Stabilized Column Generation

The rationale behind the concept presented in this subsection is the already introduced idea that a *stable* convergence of the dual variables is an indication for a *good* approximation process of the feasible region of the dual master program. A first approach was presented in our discussion of initial bases, *viz.* bounding the dual variable values. A related but more sophisticated control of the dual variables intuitively proceeds as follows.

The original motivation is to maximize a concave function over a closed convex set by considering a finite sequence of local restrictions. Adopted to our situation, we are given the optimal dual solution $\hat{\boldsymbol{u}}$ to the current dual restricted master program. By imposition of lower and upper bounds, respectively, dual variables are constrained to lie in a "box around $\hat{\boldsymbol{u}}$." The such restricted problem is re-optimized. If the new dual optimum is attained on the boundary of the box, we have a direction towards which the box should be relocated. Otherwise, the optimum is attained in the box's interior, producing the sought global optimum. This is the principle of the `Boxstep` method introduced by MARSTEN (1975) and MARSTEN, HOGAN & BLANKENSHIP (1975).

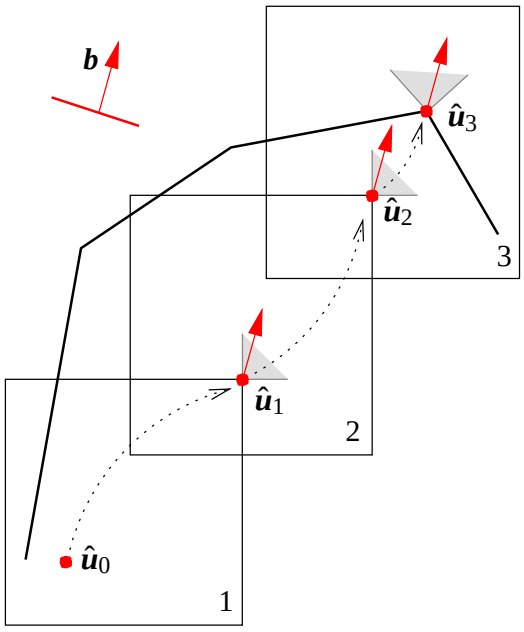

Figure 2.15: Proceeding of the `Boxstep` method in dual space

DU MERLE, VILLENEUVE, DESROSIERS & HANSEN (1999) build on these ideas, introducing the framework of *stabilized column generation*, which is an adaptation and refinement of the above. It incorporates a more flexible concept of a *box*, which entirely fits into linear programming. Therefore, it can be directly used in a column generation algorithm. Moreover, the method comprises a *perturbation* of the primal right hand side in order to overcome difficulties with degeneracy. Consider the following linear program in equality form, and convexity constraints omitted for the ease of presentation.

$$\begin{array}{rrcl} \min & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{\lambda} - \boldsymbol{\delta}_-^{\mathsf{T}}\boldsymbol{y}_- + \boldsymbol{\delta}_+^{\mathsf{T}}\boldsymbol{y}_+ & & \\ \text{subject to} & A\boldsymbol{\lambda} - \boldsymbol{y}_- + \boldsymbol{y}_+ & = & \boldsymbol{b} \\ & \boldsymbol{y}_- & \leq & \boldsymbol{\varepsilon}_- \\ & \boldsymbol{y}_+ & \leq & \boldsymbol{\varepsilon}_+ \\ & \boldsymbol{\lambda}, \boldsymbol{y}_-, \boldsymbol{y}_+ & \geq & \boldsymbol{0} \ . \end{array} \tag{2.18}$$

After changing the sign of $\boldsymbol{w}_-, \boldsymbol{w}_+$, respectively, its dual reads

$$\begin{array}{rrcl} \max & \boldsymbol{u}^\mathsf{T}\boldsymbol{b} - \boldsymbol{w}_-^\mathsf{T}\boldsymbol{\varepsilon}_- - \boldsymbol{w}_+^\mathsf{T}\boldsymbol{\varepsilon}_+ & & \\ \text{subject to} & \boldsymbol{u}^\mathsf{T}A & \leq & \boldsymbol{c}^\mathsf{T} \\ & -\boldsymbol{u}^\mathsf{T} - \boldsymbol{w}_-^\mathsf{T} & \leq & -\boldsymbol{\delta}_-^\mathsf{T} \\ & \boldsymbol{u}^\mathsf{T} - \boldsymbol{w}_+^\mathsf{T} & \leq & \boldsymbol{\delta}_+^\mathsf{T} \\ & \boldsymbol{w}_-^\mathsf{T}, \boldsymbol{w}_+^\mathsf{T} & \geq & \boldsymbol{0}^\mathsf{T} \quad . \end{array} \tag{2.19}$$

In (2.18), variables $\boldsymbol{y}_-, \boldsymbol{y}_+$ account for a perturbation of $\boldsymbol{b}$ by an amount taken from the interval $[-\boldsymbol{\varepsilon}_-, \boldsymbol{\varepsilon}_+]$. Their usage is penalized via $\boldsymbol{\delta}_-^\mathsf{T}, \boldsymbol{\delta}_+^\mathsf{T}$, respectively. The interpretation of (2.19) is more interesting. The original dual variables $\boldsymbol{u}$ are restricted to the interval $[\boldsymbol{\delta}_- - \boldsymbol{w}_-, \boldsymbol{\delta}_+ + \boldsymbol{w}_+]$, that is, deviation of $\boldsymbol{u}$ from the interval $[\boldsymbol{\delta}_-, \boldsymbol{\delta}_+]$ is penalized by an amount of $\boldsymbol{\varepsilon}_-, \boldsymbol{\varepsilon}_+$ per unit, respectively. Clearly, the motivation is to steer $\boldsymbol{u}$ towards a hopefully good estimate of the optimal $\boldsymbol{u}^\star$.

Discussion of the Stabilization Approach

Two questions immediately arise. Under what circumstances will (2.18) yield the optimal solution $\boldsymbol{\lambda}^\star$ to the unperturbed primal master program, and what are *good* choices for the parameters $\boldsymbol{\delta}_\pm, \boldsymbol{\varepsilon}_\pm$? We provide the answers in turn. Sufficient conditions for the first goal are (a) $\boldsymbol{\varepsilon}_- = \boldsymbol{\varepsilon}_+ = \boldsymbol{0}$ or (b) $\boldsymbol{\delta}_- < \boldsymbol{u}^\star < \boldsymbol{\delta}_+$, (a) for the obvious reason, and (b) by means of $\boldsymbol{\varepsilon}_\pm \geq \boldsymbol{0}$ and strong duality.

The parameters should be dealt with dynamically in the course of column generation, so as to make greatest use of the respective latest information. The above *stopping criteria* (a) and (b) act as target, towards which the parameters must develop. A final choice that does not interfere with the original master problem solution is $\boldsymbol{\delta}_-$ ($\boldsymbol{\delta}_+$) componentwise equal to $-\infty$ (∞), and $\boldsymbol{\varepsilon}_\pm$ reasonably small in each component. Straight forward *update strategies* per iteration, starting from given $\boldsymbol{\delta}_\pm^0, \boldsymbol{\varepsilon}_\pm^0$, are as follows. With intent to reduce the dual variables' variation select $\boldsymbol{\delta}_\pm$ to form a small *box* containing the current dual solution $\boldsymbol{u}$. This update could be performed in each iteration, or alternatively, each time a dual solution of currently *best quality* is obtained, which can be judged by means of the lower bounds discussed in the previous subsection. Correspondingly, the interval $[-\boldsymbol{\varepsilon}_-, \boldsymbol{\varepsilon}_+]$ could be narrowed each time a dual solution of better quality is obtained or no column with negative reduced cost coefficient is found, and otherwise could be widened. We do not intend to introduce more formalism to this update procedures due to their *heuristic* nature. It is instructive, however, to consider a particular finitely converging update strategy used by DU MERLE, VILLENEUVE, DESROSIERS & HANSEN (1999) for a set partitioning problem.

The authors define a sequence of intervals $[50, 100]$, $[0, 100]$, $[-50, \infty)$, and $(-\infty, \infty)$. Each time the column generation algorithm *stalls* the values of $\boldsymbol{\delta}_-$ and $\boldsymbol{\delta}_+$, respectively, are componentwise set to the next interval in line. For the expected optimal objective function value, the first interval is a rough estimate of the components of $\boldsymbol{u}$, *viz.* $\boldsymbol{u}^\mathsf{T}\boldsymbol{1}/m$ plus/minus a tolerance. On the contrary, $\boldsymbol{\varepsilon}_-$ and $\boldsymbol{\varepsilon}_+$ are kept fixed and are componentwise chosen at random from the intervals $[9, 11]$ and $[0, 10^{-4}]$, respectively. The first allows for an overcovering of the rows particular to the problem at hand, thus *simulating* a set covering formulation, c.f. Figure 2.9 for a discussion of the benefits of doing so. The second intervals slightly perturbs the right hand side, meant as a remedy for primal degeneracy.

From this it becomes clear that *problem specific* adaptation of the parameter choice is imperative. The preliminary experiments conducted by DU MERLE, VILLENEUVE, DESROSIERS & HANSEN (1999) indicate considerable speedup of up to a factor of ten or a growth in the size of manageable problems when using the stabilization approach. We remark that AGARWAL, MATHUR & SALKIN (1989) make similar experiences for a related proposal.

Trust Region Stabilization Device

Certainly, it is desirable to have the stabilization device independent of customization. In order to have a *direct* control of the dual variables, KALLEHAUGE (2000) considers the dual restricted master program, again with additional "box constraints" imposed on the dual variables, centered around the current dual optimal solution $\hat{\boldsymbol{u}}$, i.e., $\hat{u}_i - \delta \leq u_i \leq \hat{u}_i + \delta$, $i \in \{1,\ldots,m\}$. The method is related to the work of MADSEN (1975) in the sense that these bounds are adjusted automatically, depending on how well the dual restricted master approximates the LAGRANGIAN dual problem. This type of method is called a *trust region method*, introduced by LEVENBERG (1944) and MARQUARDT (1963). The trust region parameter δ is updated in each iteration according to the original update scheme by MARQUARDT (1963). Only iterations yielding *primal* progress are actually performed, and KELLEY's cutting plane method (1961) is applied to generate rows of the dual restricted master, i.e., columns of the primal. When the duality gap closes (up to a preset accuracy) for a dual solution in the interior of the current box, optimality is reached, and the algorithm terminates. Figure 2.14 impressively demonstrates the method's efficiency.

Weighted Dantzig-Wolfe Decomposition

The motivation above leads WENTGES (1997) to "search for good LAGRANGIAN multipliers in the neighborhood of the best multipliers found so far," when computing the LAGRANGIAN dual (2.5). More precisely, in lieu of pricing with the optimal dual variables $\boldsymbol{u}^k$ in the k^{th} iteration, the latter are convex combined as follows

$$\boldsymbol{u}^{k+1} := \frac{1}{\omega_k}\boldsymbol{u}^k + \frac{\omega_k - 1}{\omega_k}\boldsymbol{u}^{best,k} \quad , \qquad \boldsymbol{u}^{best,k} := \max_{i=1,\ldots,k} \mathcal{L}(\boldsymbol{u}^i) \quad , \tag{2.20}$$

where $\mathcal{L}(\boldsymbol{u})$ is defined in (2.4), and

$$\omega_k := \min\{const,\ (k + \text{ number of improvements of } \mathcal{L}(\boldsymbol{u}^{best,\cdot}))/2\} \text{ with } const \geq 2 \quad .$$

Obviously, (2.20) is biased towards the dual solution, which produced the respective best LAGRANGIAN lower bound in the column generation process. This emphasis becomes even stronger as the algorithms proceeds, and grows with the number of improvements of the lower bound. In a sense, this can be seen as a stabilization of heuristically good multipliers. The constant *const* is instrumental in ensuring the consideration of enough fresh information from the current restricted master program.

Rewriting (2.20) as $\boldsymbol{u}^{k+1} := \boldsymbol{u}^{best,k} + \omega_k^{-1}(\boldsymbol{u}^k - \boldsymbol{u}^{best,k})$ the method is interpreted as feasible direction search, emerging from $\boldsymbol{u}^{best,k}$ in the direction of the current dual solution $\boldsymbol{u}^k$ with step

length ω_k^{-1}. Finiteness of the weighted DANTZIG-WOLFE decomposition method is proven. Moreover, it holds that a lower bound improvement is possible for the weighted variant whenever it is for the standard method, but not vice versa. Nevertheless, the actual progress per iteration may be smaller. In computational experience with capacitated facility location problems, the weighted method succeeds in furnishing better LAGRANGIAN lower bounds, when termination is guided by a small size of the duality gap. On the other hand, the same results indicate, that the primal objective function value of the restricted master program decreases more slowly when using the new method. Nonetheless, this is an example where ideas from proposals for multiplier adjustment in subgradient methods can be transfered to the column generation context. In fact, BARAHONA & JENSEN (1998) combine the two approaches, i.e., every few iterations some or all of the dual variables obtained from the restricted master are subject to some iterations of a subgradient algorithm before being passed to the subproblem. In early iterations this produces good multipliers, later on improves the lower bound. Considerably reduced computation times are reported for their particular application. The voluminous fund of "LAGRANGIAN literature" may further furnish stimulation in this direction.

Remark. In a different line of research, DANTZIG-WOLFE type *approximation algorithms* with guaranteed convergence rates have been proposed for certain linear programs. We refer to KLEIN & YOUNG (1999), and the references given therein.

Concluding Remarks

In particular the discussion of this present section reveals that DANTZIG-WOLFE decomposition bears quite some computational defects in comparison to directly solving a linear program e.g., by the simplex method. Except that large scale models are not amenable to such a direct approach. In this respect, column generation is regarded not as a competing but as a *complementary* pricing scheme, designed to extend the range of applicability of the simplex method. All ingredients greatly benefit from customization and tailoring to the particular application, and only few things are said in this chapter about actually implementing a column generation code. We refer to the voluminous synopsis on this topic by DESAULNIERS, DESROSIERS & SOLOMON (1999), and the references given therein.

In the author's opinion, the growing understanding of the dual point of view brought considerable progress to the column generation theory and practice. It stimulated careful initializations, sophisticated solution techniques for restricted master program and subproblem, as well as better overall performance. Thus, it is an ever recurring concept in our "selected topics" of this chapter. Still, ample room for research is left, and hopefully some directions have been pointed to.

2.6 Integer Solutions

Revisiting the column generation applications listed in Table 2.1 on page 38, we conclude that, in fact, almost all of them require an *integer* solution. However, at best, solving the linear program-

ming relaxation of the respective (mixed) integer programs by column generation does provide such a solution only by chance. For the cutting stock problem it is known that, often an integral objective function value is obtained by rounding up the fractional optimum (MARCOTTE 1985). In general, however, one has to resort to some further procedure, typically branch-and-bound. It must be pointed out that this proceeding is a heuristic only, since the columns generated at the LP optimum need not contain an optimal integer solution. As a matter of fact, the corresponding variables need not even allow for an integer *feasible* solution.

2.6.1 Branch-and-Price

With a fundamental background in linear programming based branch-and-bound, *see* e.g., the book by NEMHAUSER & WOLSEY (1988), an idea for an exact algorithm to obtain integer solutions immediately comes to mind, *viz.* solving the linear programming relaxation at *each* node of the branch-and-bound tree via column generation. This method is known as *branch-and-price*, also as *integer programming column generation* (VANDERBECK & WOLSEY 1996). Excellent recent surveys are by DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995) and BARNHART, JOHNSON, NEMHAUSER, SAVELSBERGH & VANCE (1998). In view of this existing work, and in accordance to the relevance for our further proceeding in this thesis, we confine ourselves with a brief outline of the basics and pitfalls.

At first glance, branch-and-price amounts to simply integrating two well studied concepts. It is indicated already in the pioneering paper by DESROSIERS, SOUMIS & DESROCHERS (1984) that this synthesis is not straight forward. The observed tailing off effect is multiplied by having to optimize the restricted master program numerous times in the tree in order to obtain a valid lower bound in each node. Lemmata 2.9 and 2.10 together with our discussion on early termination in Subsection 2.5.3 are of help in this respect. More severe difficulties stem from the fact that a certain *harmony* between branching decisions and generator subproblems must be established so as to maintain computational tractability of the latter. The reason is easiest explained when considering a set partitioning type master program. Upward branching on a fractional variable merely reduces the subproblem and causes no trouble. Downward branching, however, explicitly forbids a particular column. Therefore, regeneration of that column must be forbidden in the pricing problem as well. This, in general, leads to finding the k^{th} best subproblem solution instead of the optimal one, *see* e.g., RIBEIRO, MINOUX & PENNA (1989) who attach the information of forbidden columns to each node of the branch-and-bound tree. Aside from the conceptual complication, we will modify or destroy a possibly exploited structure. This is all the more important when e.g., combinatorial algorithms are used for the subproblem solution.

Branch-and-Price-and-Cut

The situation gets even more involved when, in addition, cutting planes are to be added to the restricted master program in order to strengthen the lower bound in each node. This procedure is consequently called *branch-and-price-and-cut*. The pricing problem must then return the re-

spective coefficients in the present cuts as well. Moreover, the latter may be decisive in whether a column prices out favorably or not. Hence, row and column generation influence one another, and the complex mechanisms are not very well understood; except from special well behaving cases like (2.17). Nonetheless, e.g., BARNHART, BOLAND, CLARKE, JOHNSON, NEMHAUSER & SHENOI (1998), BARNHART, HANE & VANCE (2000), and PARK, KANG & PARK (1996) report on successful applications of this elaborate technique.

2.6.2 Branching Decisions

Branching in a branch-and-bound environment pursues two aims: Detect integer solutions, and provide good bounds so as to attest solution quality. A valid branching rule divides, desirably partitions, the solution space in such a way that the current fractional solution is excluded, integer solutions remain intact, and finiteness of the algorithm is ensured. Moreover, some general rules of thumb prove useful, such as to produce branches of possibly equal size, sometimes referred to as balancing the tree. Also, *important* decisions should be made early in the tree. In addition, a *compatible branching rule* is one which prevents columns that have been branched on from being regenerated without a significant complication of the pricing problem. How can all of this be achieved? JOHNSON (1989), SAVELSBERGH (1997), VANCE (1998), and VANDERBECK (1994–2000), among others, evaluate this question also from a theoretical point of view. We give just one by now classical example based on the following.

Proposition 2.11 (Branching Idea for Set Partitioning Problems, RYAN & FOSTER 1981) *Given $A \in \{0,1\}^{m\times|Q'|}$ and a fractional basic solution to $A\boldsymbol{\lambda} = \mathbf{1}$, $\boldsymbol{\lambda} \geq \mathbf{0}$, i.e., $\boldsymbol{\lambda} \notin \{0,1\}^m$. Then there exist $r,s \in \{1,\ldots,m\}$ such that $0 < \sum_{q\in Q'} a_{rq}a_{sq}\lambda_q < 1$.*

When such two rows are identified, we obtain one branch in which these rows must be covered by the same column, i.e., $\sum_{q\in Q'} a_{rq}a_{sq}\lambda_q = 1$, and one branch in which they must be covered by two distinct columns, i.e., $\sum_{q\in Q'} a_{rq}a_{sq}\lambda_q = 0$. Note, that this information can be easily transfered to and obeyed by the pricing problem.

DESROSIERS, DUMAS, SOLOMON & SOUMIS (1995) observe that if the compact formulation (CF) is solved by branch-and-bound, branching will take place on original variables. Also in LAGRANGIAN relaxation no new variables are introduced. Considering DANTZIG-WOLFE decomposition and LAGRANGIAN relaxation as equivalent primal and dual methods, respectively (*see* MAGNANTI, SHAPIRO & WAGNER 1976), this suggests that in branch-and-price, branching decisions should be derived from the original variables as well. This usually amounts to expressing the original variables in terms of a linear combination of the variables of the restricted master program, on which then bounds are imposed, *see* e.g., DESROCHERS, LENSTRA, SAVELSBERGH & SOUMIS (1991) and SOL (1994) for examples.

We have seen that what is known as variable selection in branch-and-bound codes is basically governed by pricing compatibility, and may be non trivial, whereas the selection of the next node to be processed needs no special tailoring. Ideas from standard branch-and-bound can be used, and depth-first search is a common choice. Again, we refer to the aforementioned surveys.

CHAPTER 3

Model Building: Weak and Strong Formulations

The "form" of a thing, then, says something about its limitation as well as its potentiality.
—JOSTEIN GAARDER, Sophie's World

In this chapter we present two different model formulations of the ESP. Getting a model correctly generated in an understandable form, and formulating or reformulating the model so that the problem can be solved are two different things, as was pointed out e.g., by JOHNSON (1989). Taking up this thread we will first generally comment on *content* and *form* of a model before proceeding with our concrete situation.

A mathematical formulation of a practical problem will hardly capture *all* conceivable details. There are several justifications for this assertion. First and foremost, not all details need to be known, important, or fully understood. For complex processes it may even be impossible to deliver a complete formal description. Secondly, collecting, maintaining, and processing huge amounts of detailed data need not be reasonable from an economic point of view. That is, there is a trade-off between the desired level of information to be considered and the efforts to be spent to gather this information. Often, a suitable aggregation of the data is a possibility to face this trade-off. Third, and closely related, a model may be too complex to be soluble within the state of the art. We will see that this may be remediable by means of an alternative formulation. In either case, we expect a model to be realistic enough to make the solution useful and valuable for the decision maker. Nevertheless, this is a property probably not unambiguously to judge.

In contrast, there is common consent in what regards the quality of a (mixed) integer program (MIP), which is the generic model approach to be discussed here. We consider a (mixed) integer

formulation for the same problem to be better, or *stronger*, synonymously, than another if its linear programming relaxation gives a tighter bound on the integer optimal objective function value. A variant to express this circumstance mathematically is the following.

Definition 3.1 (WOLSEY 1998)
Given a set $X \subseteq \mathbb{Z}^p \times \mathbb{R}^q$, a polyhedron P is a formulation of X if and only if $X = P \cap (\mathbb{Z}^p \times \mathbb{R}^q)$. Given two formulations P_1 and P_2 for X, P_1 is a better formulation than P_2 if $P_1 \subset P_2$.

Linear programming based branch-and-bound is a popular and outstandingly successful solution approach to combinatorial optimization problems. In this context, there is good reason for the above definition of quality. It is well known that strong LP bounds are one major ingredient to keep the search tree manageable. We emphasize, however, that other exact and heuristic methods are available in the literature, and each of these may benefit from individual considerations we do not cover here.

3.1 Mixed Integer Formulation

Lemma 1.7 states strong $\mathcal{NP}$-completeness as computational complexity status of the ESP. That is, there is strong evidence that no pseudo-polynomial, let alone efficient, algorithms for its solution exist. A formulation as an integer program is an appropriate and well established means to expose and study the structure of our problem.

The traditional mixed integer formulation of the related m-PDPTW (DUMAS, DESROSIERS & SOUMIS 1991, SOL 1994) involves explicit decisions on assignments of requests to vehicles, the sequence of visited pickup and delivery locations, vehicle arrival times, and vehicle capacities at each location. We refrain from repeating this mixed integer program here and proceed directly to a version adapted to our special situation.

Let us recall the ESP data collected in Table 1.1 and the associated request graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ as introduced in Section 1.3. Some additional notation and definitions will be necessary. Given a pattern $P = \{i_1, i_2, \ldots, i_k\} \in \mathcal{P}$, $\mathcal{A}|_P \subseteq \mathcal{A}$ denotes the restriction of $\mathcal{A}$ to the arcs within P, i.e., $\mathcal{A}|_P = \{(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)\}$. Moreover, $o(P) = i_1$ and $d(P) = i_k$ are used for the *origin* and the *destination*, respectively, of the pattern P. In particular, $o(\{e^-\}) := e^-$ and $d(\{e^+\}) := e^+$. For any $e \in \mathcal{E}$ we define the service times $s_{e^+} = s_{e^-} = 0$. For a pattern $P \in \mathcal{P}$, $\boldsymbol{\delta}_P$ denotes its request incidence vector, the r^{th} component δ_{rP}, $r \in \mathcal{R}$, of which equals to one if $r^+, r^- \in P$, and to zero otherwise. Finally, we denote the respective subset of patterns for which engine $e \in \mathcal{E}$ is admissible by $\mathcal{P}_e \subseteq \mathcal{P}$. We now introduce the variables:

z_P^e	one if pattern $P \in \mathcal{P}_e$ is assigned to engine $e \in \mathcal{E}$; zero otherwise
$x_{P_1P_2}^e$	one if pattern $P_2 \in \mathcal{P}_e$ is immediately served after $P_1 \in \mathcal{P}_e$ on engine $e \in \mathcal{E}$; zero otherwise
T_i	(non-negative) arrival time in location $i \in \mathcal{N}$ of the visiting engine

Note, that for the arrival times we need not specify *which* engine arrives at a particular location since this information is already covered by the z variables. Table 3.1 shows the entire formulation (3.1) through (3.10) which we denote by (ESPMIP).

$$\text{minimize} \quad \sum_{e\in\mathcal{E}} T_{e^-} \tag{3.1}$$

$$\text{subject to} \quad \sum_{e\in\mathcal{E}}\sum_{P\in\mathcal{P}_e} \delta_{rP} z^e_P = 1 \qquad \forall r\in\mathcal{R} \tag{3.2}$$

$$\sum_{P\neq P_1\in\mathcal{P}_e\cup\{e^+\}} x^e_{P_1P} = \sum_{P\neq P_2\in\mathcal{P}_e\cup\{e^-\}} x^e_{PP_2} = z^e_P \qquad \forall e\in\mathcal{E}, P\in\mathcal{P}_e \tag{3.3}$$

$$\sum_{P\in\mathcal{P}_e\cup\{e^-\}} x^e_{\{e^+\}P} = 1 \qquad \forall e\in\mathcal{E} \tag{3.4}$$

$$\sum_{P\in\mathcal{P}_e\cup\{e^+\}} x^e_{P\{e^-\}} = 1 \qquad \forall e\in\mathcal{E} \tag{3.5}$$

$$z^e_P = 1 \;\Rightarrow\; T_i + s_i + t_{ij} \le T_j \qquad \forall e\in\mathcal{E}, P\in\mathcal{P}_e, (i,j)\in\mathcal{A}|_P \tag{3.6}$$

$$x^e_{P_1P_2} = 1 \;\Rightarrow\; T_{d(P_1)} + s_{d(P_1)} + t_{d(P_1)o(P_2)} \le T_{o(P_2)} \qquad \forall e\in\mathcal{E}, P_1\neq P_2\in\mathcal{P}_e\cup\dot{\bigcup}_{e\in\mathcal{E}}\{e^+,e^-\} \tag{3.7}$$

$$\underline{t}_i \le T_i \le \bar{t}_i \qquad \forall i\in\mathcal{N} \tag{3.8}$$

$$z^e_P \in \{0,1\} \qquad \forall e\in\mathcal{E}, P\in\mathcal{P}_e \tag{3.9}$$

$$x^e_{P_1P_2} \in \{0,1\} \qquad \forall e\in\mathcal{E}, P_1\neq P_2\in\mathcal{P}_e\cup\dot{\bigcup}_{e\in\mathcal{E}}\{e^+,e^-\} \tag{3.10}$$

Table 3.1: The mixed integer formulation (ESPMIP) for the engine scheduling problem

The given objective function (3.1) serves as example only and aims at an earliest possible completion time for each engine. When, for instance, the primary objective is to minimize the *fleet size*, an alternative to (3.1) would look like

$$\text{minimize} \quad \sum_{e\in\mathcal{E}}\sum_{P\in\mathcal{P}_e} M_e\cdot x^e_{\{e^+\}P} + \sum_{e\in\mathcal{E}} T_{e^-} \quad , \tag{3.11}$$

where M_e represents the (comparatively large) fixed cost of operating engine $e\in\mathcal{E}$. In the case that completion times are of less importance, another variant is to focus on the *total travel cost* (or distance), i.e.,

$$\text{minimize} \quad \sum_{e\in\mathcal{E}}\sum_{P\in\mathcal{P}_e}\sum_{(i,j)\in\mathcal{A}|_P} c_{ij}\cdot z^e_P + \sum_{e\in\mathcal{E}}\sum_{P_1\neq P_2\in\mathcal{P}_e\cup\{e^+,e^-\}} c_{d(P_1)o(P_2)}\cdot x^e_{P_1P_2} \quad . \tag{3.12}$$

Let us now turn to the interpretation of the constraints. We ensure by (3.2) that the selection of patterns indeed partitions $\mathcal{R}$. An engine must visit and leave precisely those patterns assigned to it, a requirement guaranteed by (3.3). By constraints (3.4) and (3.5) each $\mathcal{P}$-concatenation

starts and ends in the appropriate locations for the relevant engine. Note that this formulation allows an engine to stay idle. The next two constraints take care of setting the visiting times of each particular location, (3.6) within a selected pattern and (3.7) in between selected patterns. Time windows for the start-of-service time are respected by (3.8). Finally, (3.9) and (3.10) define the domains of the binary variables.

What immediately meets the eye when we compare (ESPMIP) to the traditional mixed integer formulation of the *m*-PDPTW mentioned above is the absence of precedence and capacity constraints. This remarkable feature is of course a consequence of the properties of patterns, i.e., of Lemma 1.3. The prize we have to pay for this simplification is that the size of our formulation directly depends on $|\mathcal{P}|$. More precisely, we have

$$\begin{aligned} O(|\mathcal{R}|) &\quad \text{continuous variables,} \\ O(|\mathcal{E}||\mathcal{P}| + |\mathcal{E}|^2|\mathcal{P}|^2) &\quad \text{binary variables,} \\ O(|\mathcal{R}| + |\mathcal{E}| + |\mathcal{P}||\mathcal{E}|(1 + |\mathcal{P}||\mathcal{E}| + |\mathcal{R}|^2)) &\quad \text{constraints.} \end{aligned}$$

Still, for our particular situation of $\mathcal{P} \equiv \mathcal{P}^{1\cup 2}$, the size of (ESPMIP) is polynomial in $|\mathcal{E}|$ and $|\mathcal{R}|$, since $|\mathcal{P}| = O(|\mathcal{R}|^2)$. The non-linear constraints (3.6) (and similarly (3.7)) are usually rewritten as linear constraints

$$T_i + s_i + t_{ij} - T_j \le M_{ij}(1 - z_P^e) \qquad \forall e \in \mathcal{E}, P \in \mathcal{P}, (i,j) \in \mathcal{A}|_P \ , \tag{3.6'}$$

where M_{ij} is a sufficiently large constant, e.g., $M_{ij} = \bar{t}_i + s_i + t_{ij} - \underline{t}_j$. Then, for $z_P^e = 0$ the constraints (3.6′) (and similarly for $x^e_{P_1P_2} = 0$ in an analogously rewritten (3.7′)) become redundant. With respect to this property, our choice of M_{ij} obviously is the smallest feasible. We will see that even this strongest possible version of the inequality is disastrous for the linear programming relaxation (ESPMIP′) of (ESPMIP), where constraints (3.9) and (3.10) are replaced by

$$z_P^e \in [0,1] \quad \forall e \in \mathcal{E}, P \in \mathcal{P}_e \tag{3.9'}$$

$$x^e_{P_1P_2} \in [0,1] \quad \forall e \in \mathcal{E}, P_1 \ne P_2 \in \mathcal{P}_e \cup \dot{\bigcup}_{e\in\mathcal{E}} \{e^+, e^-\} \ . \tag{3.10'}$$

Lemma 3.2 (Lower Bound Obtained from (ESPMIP′))
Let (ESPMIP) have a feasible solution. Let $\mathcal{P}_e = \mathcal{P}$, $e \in \mathcal{E}$. For $M_{ij} \ge (\bar{t}_i + s_i + t_{ij} - \underline{t}_j) \cdot |\mathcal{E}|/(|\mathcal{E}| - 1)$ in (3.6′) and (3.7′) we have

$$z^\star_{\text{ESPMIP}'} = \sum_{e\in\mathcal{E}} \underline{t}_{e^-} \ .$$

Proof. Let the ordered sets $\{\{e^+\}, P_1^e, \ldots, P_{\nu_e}^e, \{e^-\}\}$, $e \in \mathcal{E}$, represent a set of $\mathcal{P}$-concatenations which form a feasible solution to the ESP. We define

$$z_P^E = \begin{cases} \frac{1}{|\mathcal{E}|} & E \in \mathcal{E}, P \in \dot{\bigcup}_{e\in\mathcal{E}} \{P_1^e, \ldots, P_{\nu_e}^e\} \\ 0 & \text{otherwise} \end{cases}$$

$$x^E_{P_i^e P_{i+1}^e} = \begin{cases} \frac{1}{|\mathcal{E}|} & E \in \mathcal{E}, e \in \mathcal{E}, i \in \{0, \ldots, \nu_e\}, \text{ where } P_0^e = \{E^+\} \text{ and } P^e_{\nu_e+1} = \{E^-\} \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that this evaluation of variables satisfies (3.2) – (3.5) since, informally, all engines "partially" serve each $\mathcal{P}$-concatenation, i.e., by a fraction of $|\mathcal{E}|^{-1}$. Now consider the constraints (3.6′) (and (3.7′) similarly) which take the form

$$T_i + s_i + t_{ij} \leq M_{ij}\left(1 - \frac{1}{|\mathcal{E}|}\right) + T_j, \quad P \in \bigcup\!\!\cdot_{e\in\mathcal{E}} \{P_1^e, \ldots, P_{\nu_e}^e\}, (i,j) \in \mathcal{A}|_P \ ,$$

yielding the redundant constraints $\underline{t}_j \leq T_j$, $j \in \mathcal{N}$, when M_{ij} is chosen as stated, and the claimed optimal objective function value $z^\star_{\text{ESPMIP}'}$ follows from the fact that all T variables, especially T_{e^-}, $e \in \mathcal{E}$, assume their respective lower bounds. □

In other words, under the (somewhat technical) assumption of Lemma 3.2 the linear relaxation of (ESPMIP) gives the *trivial* lower bound. Worse than that, however, when time windows are not tight (which is the case for our ESP instances) it is likely to have $T_i < \bar{t}_i$, $i \in \mathcal{N}$, in the optimal solution of (ESPMIP′). That is, we may drop the lemma's assumption on M_{ij}, and obtain no better bound than the trivial one even for the strongest possible formulation of (3.6′) and (3.7′). It is thus worthless as initial bound for a branch-and-bound procedure. In the next section we will see that the situation is still worse; the bound needs not increase even after quite a while of branching. When the assumption $\mathcal{P}_e = \mathcal{P}$, $e \in \mathcal{E}$ does not hold, the z variables are chosen such that each served pattern is distributed evenly among the admissible engines only. The proof's argument then is similar, although this may improve the quality of the lower bound under the condition that time windows are sufficiently tight.

3.1.1 Problem Symmetry

Before proceeding, we would like to point out a potential defect of a (mixed) integer program which becomes apparent only when also the *solution method* is taken into consideration. Problem instances have been reported upon on which linear programming based branch-and-bound "runs forever" (JOHNSON 1989). Besides the already mentioned—and to the utmost important—quality of the lower bound itself there is a strongly related reason for such a poor performance. A circumscription taken from the literature is that variables do not reflect *important* partial decisions in a complex decision problem.

In the formulation (ESPMIP) for the ESP, when it comes e.g., to branching on a fractional x variable, deciding that the corresponding sequence of patterns will not be incorporated in a concatenation for a particular engine may have very little effect on the linear programming optimum. An explanation is as follows. It is well possible that two (or more) concatenations $\{P_1^1, \ldots, P_{\nu_1}^1\}$ and $\{P_1^2, \ldots, P_{\nu_2}^2\}$ are admissible for two (or more) different engines $e_1, e_2 \in \mathcal{E}$. From the proof of Lemma 3.2 we conclude that it is likely that each engine's service is fractionally split over several concatenations. Assume for simplicity that for the above two concatenations the relevant variables assume, say,

$$x^{e_k}_{P_i^k P_{i+1}^k} = 0.5, \ \ i = 1, \ldots, \nu_k - 1, \ k \in \{1,2\} \ .$$

Branching downwards on one of these variables, i.e., fixing a variable to zero, may result e.g., in a mere change of rôles of the two engines, i.e.,

$$x^{e_1}_{P_i^2 P_{i+1}^2} = 0.5, \;\; i = 1, \ldots, \nu_2 - 1, \quad \text{and} \quad x^{e_2}_{P_i^1 P_{i+1}^1} = 0.5, \;\; i = 1, \ldots, \nu_1 - 1 \;\; .$$

SHERALI & SMITH (2000) call this the *reindexing* of indistinguishable objects. The T variables are clearly not affected by this operation, neither so is the optimal objective function value.

More complicated scenarios are conceivable e.g., many more than two concatenations may be admissible for different engines, possibly leading to a very large number of alternative solutions to the *model* which do not differ significantly in their meaning for the original *problem*. In the context of branch-and-bound the consequence is that even repeated branching does not necessarily improve the quality of the LP bound. Probably worse from a computational standpoint, branching creates, in a sense, *symmetric* subtrees, which must be all explored in more or less the same way. Each subtree will provide large amounts of redundant information in what regards bounds and feasible solutions.

When identical or similar solutions are reproducible in many different ways one refers to *problem symmetry*. Note again, that this is an attribute of the formulation not of the problem at hand.[1] Several authors hinted at this problem (*see* e.g., JOHNSON 1989, JOHNSON, MEHROTRA & NEMHAUSER 1993, BARNHART, HANE & VANCE 2000, VANCE 1998), and some suggested alternative formulations with fewer symmetry—a matter we will follow up in the next section.

We finally remark that when the cause of symmetry can be identified, the introduction of symmetry-breaking hierarchical constraints can be a remedy to enforce unique representations of distinct solutions (SHERALI & SMITH 2000, SMITH 2000). The mixed integer program `noswot` taken from the MIPLIB[2] library is a demonstration of the computational impact of such a constraint (BIXBY 2000*a*). We did not experiment with hierarchical constraints for (ESPMIP), in favor of an alternative, stronger formulation to be presented next.

3.2 Set Partitioning Formulation

Our definition of a pattern family already reflects the fact that every feasible solution to the ESP partitions the set of requests, c.f. Definition 1.2 on page 17. Since every request is to be covered by exactly one $\mathcal{P}$-concatenation it is a natural idea to base decisions on entire concatenations, resulting in a set partitioning formulation, c.f. Subsection 2.1.3. Denote by Ω_e the set of *all* subsets $R \subseteq \mathcal{R}$ having the property that a feasible $\mathcal{P}^{1\cup 2}$-concatenation for engine $e \in \mathcal{E}$ visiting exactly all $r \in R$ exists. The elements of Ω_e will be called the *admissible request sets* for engine $e \in \mathcal{E}$. Associated with each $R \in \Omega_e$ is an incidence vector $\boldsymbol{\delta}_R$ whose r^{th} component δ_{rR} equals to one if $r \in R$, and zero otherwise. Ordinarily, given $R \in \mathcal{R}$, there exist many different feasible

[1] The notion of *model symmetry* appears to reflect matters more appropriately.

[2] *See* `http://www.crpc.rice.edu/softlib/catalog/miplib.html`. MIPLIB is an electronic library of both pure and mixed integer programming problems created in an effort to make real models available to researchers working in this area.

concatenations visiting exactly all $r \in R$. However, we do not lose any generality when we assign to each $\boldsymbol{\delta}_R$ the smallest possible cost coefficient c_R^e with respect to the given cost evaluation of concatenations for engine $e \in \mathcal{E}$. Identical columns with larger cost will not be part of an optimal selection. Thus, we obtain a way to reconstruct a concatenation from its incidence vector. Whether a particular $R \subseteq \Omega_e$ is selected or not is represented by a binary variable λ_R^e. To summarize, this model reads as follows:

$$\begin{aligned} \min \quad & \sum_{e\in\mathcal{E}} \sum_{R\in\Omega_e} c_R^e \lambda_R^e & & \\ \text{subject to} \quad & \sum_{e\in\mathcal{E}} \sum_{R\in\Omega_e} \delta_{rR} \lambda_R^e &= 1, \qquad & r \in \mathcal{R} \\ & \sum_{R\in\Omega_e} \lambda_R^e &\leq 1, \qquad & e \in \mathcal{E} \\ & \lambda_R^e &\in \{0,1\}, \quad & e \in \mathcal{E},\, R \in \Omega_e \ . \end{aligned} \tag{ESPSP}$$

In other words, choose for each engine $e \in \mathcal{E}$ at most one admissible request set $R \in \Omega_e$ such that the disjoint union of chosen sets precisely yields $\mathcal{R}$ at a minimum cost. This seemingly compact formulation has some serious detriments. At first, note, that determining the cost coefficient of a particular variable itself is an $\mathcal{NP}$-complete combinatorial optimization problem since it involves the solution of a 1-ESP, c.f. Lemma 1.7. Secondly, although the model has very few constraints, in general each Ω_e is of exponential cardinality, resulting in prohibitively many variables. Thus even the solution of the linear programming relaxation (ESPSP$'$) in a straightforward manner is impracticable. Despite this discouraging evidence, the column generation approach introduced in Chapter 2 enables us to cope with this latter difficulty. The details are postponed to Chapter 4.

We will now see that the set partitioning formulation outperforms the mixed integer formulation in terms of the quality of the lower bound provided by the linear programming relaxation. That is, in the sense defined above, (ESPSP) is a *better formulation* for the ESP.

Lemma 3.3 $z^\star_{\text{ESPMIP}'} \leq z^\star_{\text{ESPSP}'}$

Proof. We perform a DANTZIG-WOLFE decomposition on (ESPMIP$'$) along the lines of Subsection 2.1.1. All variables are bounded, thus $X_e = \{(\boldsymbol{z}^e, \boldsymbol{x}, \boldsymbol{T}) \mid (3.3) - (3.8), (3.9'), (3.10')\}$ is a polytope for any fixed $e \in \mathcal{E}$. Denoting the set of extreme points of X_e by $\{(\overline{\boldsymbol{z}}_q^e, \overline{\boldsymbol{x}}_q, \overline{\boldsymbol{T}}_q)\}_{q\in Q_e}$, with a finite index set Q_e, we obtain a formulation equivalent to (ESPMIP$'$):

$$\begin{aligned} \min \quad & \sum_{e\in\mathcal{E}} \sum_{q\in Q_e} \overline{T}_{qe^-} \cdot \mu_q^e & & \\ \text{subject to} \quad & \sum_{e\in\mathcal{E}} \sum_{P\in\mathcal{P}_e} \sum_{q\in Q_e} \delta_{rP} \cdot \overline{z}_{qP}^e \cdot \mu_q^e &= 1, \quad & r \in \mathcal{R} \\ & \sum_{q\in Q_e} \mu_q^e &= 1, \quad & e \in \mathcal{E} \\ & \mu_q^e &\geq 0, \quad & e \in \mathcal{E},\, q \in Q_e \ . \end{aligned}$$

Now let $R \subseteq \mathcal{R}$ be a (possibly empty) subset of requests admissible for engine $e \in \mathcal{E}$, i.e., $R \in \Omega_e$. Then there exists a collection of patterns $\tilde{\mathcal{P}}_e \subseteq \mathcal{P}_e$ such that $\sum_{P \in \tilde{\mathcal{P}}_e} \boldsymbol{\delta}_P = \boldsymbol{\delta}_R$ or, more specifically, there is a $(\tilde{\boldsymbol{z}}^e, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{T}}) \in X_e$ such that

$$\sum_{P \in \mathcal{P}_e} \boldsymbol{\delta}_P \tilde{z}^e_P = \boldsymbol{\delta}_R \qquad \text{and therefore} \qquad \sum_{P \in \mathcal{P}_e} \boldsymbol{\delta}_P \sum_{q \in \bar{Q}_e} \bar{z}^e_{qP} \cdot \hat{\mu}^e_q = \boldsymbol{\delta}_R, \quad \sum_{q \in \bar{Q}_e} \hat{\mu}^e_q = 1$$

for some $\bar{Q}_e \subseteq Q_e$. In other words, each solution to (ESPSP′) is feasible for (ESPMIP′). Hence, (ESPMIP′) is a relaxation of (ESPSP′). Moreover, with $c^e_R := \sum_{q \in \bar{Q}_e} \bar{T}_{qe^-} \cdot \hat{\mu}^e_q$ the objective function value is preserved for a solution, and the claim follows. □

This result complies to our investigations in Chapter 2 and has been previously observed by several authors in similar contexts, *see* GEOFFRION (1974) for the original reference. Note, that for non-integral $\tilde{\boldsymbol{z}}^e$, and therefore non-integral $\hat{\mu}^e_q$, strict inequality is possible in Lemma 3.3. Not only that (ESPSP′) provides a stronger lower bound, but also, it is "folklore" that the linear relaxation of a set partitioning problem usually is very tight. Often, the linear programming optimum is already integral, though not necessarily optimal. An explanation for this behavior is that each integral solution to a set partitioning problem is a basic solution of the associated linear relaxation (BALAS & PADBERG 1972, 1975).

In what regards problem symmetry we make two observations, *viz.* (a) the very same request set may be admissible for multiple engines, however, the associated columns need not represent the same concatenation and thus may have distinct cost coefficients. Even if (b) these concatenations are identical, there are at most $|\mathcal{E}|$ identical columns, since we postulated to consider for each engine only a cheapest possible concatenation per given request set. Compared to (ESPMIP) there is much less combinatorial freedom for obtaining structurally identical solutions (of the same cost) to the ESP. Together with Lemma 3.3 these theoretical advantages suggest (ESPSP) as the more appropriate formulation to solve practical instances.

Remark. For the related but somewhat simpler *m*-VRPTW BRAMEL & SIMCHI-LEVI (1997) show that for a set covering formulation and for any distribution of service times, time windows, locations, and customer loads, the relative integrality gap becomes arbitrarily small as the number of customers increases. Moreover, they prove that an edge oriented alternative formulation has an asymptotic relative gap of 50% at the worst.

3.3 Model Improvements and Extensions

Mixed integer programming offers great flexibility in setting up a model for a particular problem, a property for which the MIP approach earned the reputation of being extraordinarily widely applicable. On the other hand, as WILLIAMS (1999) puts it, this "results in a greater divergence between good and bad models of a practical problem"—much in contrast to linear programming. However, we have emphasized that judging the quality of a model is sensible in conjunction with

the method used for its solution only. For branch-and-bound methods the tightness of the lower bound (minimization assumed) is crucial, and modeling efforts should respect this.

The state-of-the-art in solving integer programming problems involves several *preprocessing* techniques such as bound strengthening, variable fixing, coefficient reduction, and eliminating redundancies. It should be pointed out clearly that these techniques substantially contribute to the solubility of models at all. Furthermore, commercial solvers like CPLEX offer the systematic addition of constraints, i.e., cutting planes, during the branch-and-bound process. One attempts that the *default settings* of e.g., CPLEX provide an acceptable solver performance on the average application (BIXBY 2000*a*). This way, deficiencies in model building are partially compensated. Nevertheless, besides these generic improvements, problem specific knowledge should be exploited whenever (efficiently) available.

Shrinking Time Windows and Network Reduction

The lower bound obtained from (ESPMIP) benefits from tight time windows (Lemma 3.2). A standard preprocessing procedure found e.g., in DUMAS, DESROSIERS & SOUMIS (1991) succesively redefines ("shrinks") the time windows as follows:

$$\bar{t}_{r^-} = \min\{\bar{t}_{r^-},\, \max_{e\in\mathcal{E}_r}\{\bar{t}_{e^-} - t_{r^-e^-}\}\} \quad \text{and} \quad \bar{t}_{r^+} = \min\{\bar{t}_{r^+},\, \bar{t}_{r^-} - t_{r^+r^-}\} \quad \forall r \in \mathcal{R} \tag{3.13}$$

$$\underline{t}_{r^+} = \max\{\underline{t}_{r^+},\, \min_{e\in\mathcal{E}_r}\{\underline{t}_{e^+} + t_{e^+r^+}\}\} \quad \text{and} \quad \underline{t}_{r^-} = \max\{\underline{t}_{r^-},\, \underline{t}_{r^+} + t_{r^+r^-}\} \quad \forall r \in \mathcal{R} \tag{3.14}$$

In the PDPTW we are enabled to delete arcs of the request graph $\mathcal{G}$ by *a priori* violated pairing, precedence, and capacity constraints (*see* DUMAS, DESROSIERS & SOUMIS 1991). As we have remarked earlier, these constraints are always satisfied for $\mathcal{P}$-concatenations. Still, time windows can be used to eliminate arc $(i,j) \in \mathcal{A}$ if $\underline{t}_i + s_i + t_{ij} > \bar{t}_j$, as a preliminary step. Note, that a side effect of this network reduction is a possible decrease in the number of admissible patterns. More preprocessing is discussed in Section 4.2.

Stronger Reformulations

Polyhedral studies of the convex hull of integer solutions of combinatorial optimization problems lead to a remarkable success of branch-and-cut approaches, where (strong) valid inequalities are added in each node of the branch-and-bound tree in order to strengthen the linear programming relaxation, *see* HOFFMAN & PADBERG (1985) for a classical reference. A special case is to add cutting planes in the root node only, a procedure we informally refer to as *cut-and-branch*.

We basically identified two sources of known valid inequalities to be employed in the context of (ESPMIP). Cutting planes for the set partitioning problem—formed by constraints (3.2)—like *clique inequalities* (PADBERG 1973) can be imposed on the z variables. Secondly, inequalities arising from the TSP like *subtour elimination constraints* of various kinds have been adapted, e.g., to the capacitated VRP, *see* CORNUÉJOLS & HARCHE (1993). RULAND & RODIN (1997)

additionally derived valid inequalities for the 1-PDP from precedence constraints. None of these approaches so far succeeded in solving instances of practical size exactly.

To date, these stronger formulations cannot keep up with the above stronger *re*formulation as a set partitioning problem. The great popularity of this latter approach, c.f. Table 2.1, is an indicator that this observation is true for many other applications also. We therefore make a point of the importance not only of the set of constraints but also of the definition of the variables. In particular, it is too myopic to consider the size of a formulation as a decisive criterion for model quality.

Model Extensions

Having discussed the two formulations in terms of the strength of the bounds obtained by their respective linear programming relaxations, we now turn again to our practical situation. The two models are not equally suited for the incorporation of additional information. All *network based* constraints, i.e., related to locations or particular tracks, are relatively easily represented in (ESPMIP). Consider, for instance, the requirement that request i must precede request j in every feasible solution, i.e., i must be entirely completed before service of j may start. This can be taken care of by

$$T_{i^-} + s_{i^-} \leq T_{j^+} \quad . \tag{3.15}$$

Moreover, one may wish to avoid congestion effects on the tracks, i.e., engines should not meet at the same location or track. When the service of two pairs (i, j) and (i', j') of locations requires traveling along the same track segment one ensures a smooth operation by introduction of a safety distance d_s in minutes. This can be accomplished e.g., by $|(T_i + s_i) - (T_{i'} + s_{i'})| \geq d_s$. Using binary variables $w_{ii'}$ indicating whether i is visited before i' ($w_{ii'} = 1$) or not ($w_{ii'} = 0$), this can be written as before, *viz.*

$$(T_i + s_i) + d_s - (T_{i'} + s_{i'}) \leq M \cdot (1 - w_{ii'}) \tag{3.16}$$

$$(T_{i'} + s_{i'}) + d_s - (T_i + s_i) \leq M \cdot w_{ii'} \tag{3.17}$$

$$w_{ii'} \in \{0, 1\} \quad , \tag{3.18}$$

where again M is a sufficiently large constant, chosen by analogy to the discussion above. Note, that in our *pattern oriented* formulation it is not directly possible to incorporate constraints which are more detailed in what regards the resolution of the track network.

The set partitioning formulation (ESPSP) easily enables us to apply restrictions on single concatenations, their composition or maximal shift lengths, for instance. These constraints are incorporated in the subproblem. Also, a precedence relation among requests which must definitively be served on the same engine can be respected this way. On the other hand, (ESPSP) is not suited for imposing constraints on the global structure of a solution, i.e., on the totality of selected concatenations. The reason is, that different admissible request sets are mutually independent in the model, apart from the partitioning requirement. That is, with (ESPSP) in its present form, we are not able to control e.g., a precedence relationship between two requests that are possibly served on *two different* concatenations. An immediate modification would be to let

variables define *entire* solutions to the ESP in the sense of Definition 1.6—an obviously absurd formulation which directly resulted in a complete enumeration.

Handling of Precedence Relations in (ESPSP)

Although the subject is of current practical interest, the literature on incorporating dependencies between different vehicles in a formulation in the style of (ESPSP) is scant. We follow the lines of IOACHIM, DESROSIERS, SOUMIS & BÉLANGER (1999), and refer to DESAULNIERS, DESROSIERS, IOACHIM, SOLOMON, SOUMIS & VILLENEUVE (1998) for a general approach.

Complementary to the information about the set $R \in \Omega$ of visited requests we may store the arrival time T_{iR} in the respective locations $i \in \mathcal{N}$. Then, we may calculate the actually realized arrival time in i as $T_i = \sum_{e\in\mathcal{E}} \sum_{R\in\Omega_e} T_{iR}\lambda_R^e$, where we postulate $T_{iR} = 0$ if location $i \in \mathcal{N}$ is not visited by R. Now suppose we are given a set Π of pairs of locations, such that $(i^-, j^+) \in \Pi$ is tantamount to the precedence relation (3.15). We would enlarge (ESPSP) in the following way.

$$
\begin{array}{rrcll}
\min & \displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega_e} c_R^e\lambda_R^e & & & \\
\text{subject to} & \displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega_e} \delta_{rR}\lambda_R^e & = & 1, & r \in \mathcal{R} \\
\displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega_e} T_{iR}\lambda_R^e - & \displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega_e} T_{jR}\lambda_R^e & \geq & s_i, & (i,j) \in \Pi \\
& \displaystyle\sum_{R\in\Omega_e} \lambda_R^e & \leq & 1, & e \in \mathcal{E} \\
& \lambda_R^e & \in & \{0,1\}, & e \in \mathcal{E},\ R \in \Omega_e\ .
\end{array}
\tag{3.19}
$$

Computing the values T_{iR} means no extra effort, since they are a by-product of determining the admissibility of $R \subseteq \mathcal{R}$ anyway which is required to set up the model in the first place. Thus, additional constraints involving the familiar time variables from the mixed integer formulation (ESPMIP) can be respected in this formulation as well. The price we have to pay for this modeling power becomes more visible when we again take into account the solution methodology, which involves the dynamic generation of the columns of the model.

Each of the new constraints introduces a new dual variable $w_{ij} \in \mathbb{R}_+$, $(i,j) \in \Pi$. Therefore, the subproblem to generate a new column, c.f. Section 2.4, must accommodate in its objective function the additional summand $-\sum_{(i,j)\in\Pi}(T_{iR} - T_{jR}) \cdot w_{ij}$. That is, we incur a cost in each location which is linear in the arrival time in that location. IOACHIM, GÉLINAS, SOUMIS & DESROSIERS (1998) and DESAULNIERS & VILLENEUVE (2000) show how to handle linear node or waiting costs in shortest path dynamic programming algorithms.

Model (3.19) implies another, much more severe difficulty. In order to guarantee a feasible solution we may have to waive the condition that a variable λ_R^e represents a *minimal cost* concatenation which visits R with engine e. Again, there are consequences for the pricing subproblem.

When a more expensive column with appropriate time variables needs to be generated this information must be made available to the subproblem. We are not aware of a more efficient way than imposing dynamically adapted bounds on the T_{iR}, which could drastically complicate solution procedures. Still, we hope that our discussion stimulates future research efforts in this direction.

We emphasize the incorporation of precedence relations into our models because this is a conceivable future extension, increasing the level of detail in which practical situations can be reflected. At the time being, however, no practical data is available to evaluate such an extension. This is due to the fact that, today, engines are scheduled rather independently from one another. Therefore, we do not further consider precedence constraints in our models.

CHAPTER 4

Engine Scheduling by Column Generation

But it would take years, perhaps decades, and we are a little impatient.
—CARL SAGAN, Contact

In this chapter we propose solving the linear programming relaxation of the set partitioning formulation (ESPSP) of the ESP by column generation. Among others, a label correcting algorithm together with a new label elimination technique will be developed for the solution of the pricing problem.

4.1 Restricted Master Program

As remarked earlier, the tremendous number of variables of (ESPSP)—emerging from the number of all admissible request sets—even prevents us from solving its linear programming relaxation directly. Instead, we will work with a small portion of the model, adjoining variables *as and when needed*. Given a subset $\Omega'_e \subseteq \Omega_e$ of admissible request sets for each engine $e \in \mathcal{E}$, the restricted master program is

$$\begin{aligned} \min \quad & \sum_{e\in\mathcal{E}} \sum_{R\in\Omega'_e} c_R^e \lambda_R^e & & \\ \text{subject to} \quad & \sum_{e\in\mathcal{E}} \sum_{R\in\Omega'_e} \delta_{rR} \lambda_R^e &=& \ 1, \quad r \in \mathcal{R} \\ & \sum_{R\in\Omega'_e} \lambda_R^e &\leq& \ 1, \quad e \in \mathcal{E} \\ & \lambda_R^e &\geq& \ 0, \quad e \in \mathcal{E}, R \in \Omega'_e \ , \end{aligned} \tag{RMP$'$}$$

where the upper bound of one on the variables is already implied by the partitioning constraints.

Initially, $\Omega' := \bigcup_{e\in\mathcal{E}} \Omega'_e$ can be thought of being *any* collection of admissible request sets that allow a partition of $\mathcal{R}$. We discuss the initialization of Ω' in the next subsection. Associated with a primal optimal solution $\boldsymbol{\lambda}^\star \in \mathbb{R}_+^{|\Omega'|}$ to (RMP′) is a dual optimal solution $(\boldsymbol{u}^\star, \boldsymbol{v}^\star)^\mathsf{T}$, where again we henceforth drop the superscript star for notational simplicity. The dual variables u_r, $r \in \mathcal{R}$, relate to the partitioning equalities and since are not restricted in sign, whereas the non-positive dual variables v_e, $e \in \mathcal{E}$, correspond to the convexity constraints. From the reduced cost coefficients $c_R^e - \boldsymbol{u}^\mathsf{T} \cdot \boldsymbol{\delta}_R - v_e$, $e \in \mathcal{E}$, $R \in \Omega_e$ we obtain the pricing problem

$$\min \left\{ c_R^e - \sum_{r\in R} u_r - v_e \mid e \in \mathcal{E}, R \in \Omega_e \right\} \quad , \tag{4.1}$$

which has to be solved in each iteration of the column generation process. We refer to Section 2.2 for an elaborate presentation of methodological details.

4.1.1 Initialization

Due to the presence of convexity constraints it is non-trivial to provide an initially feasible collection of admissible request sets. More precisely, from Lemma 1.8 we know that finding initial sets $\Omega'_e \subseteq \Omega_e$, $e \in \mathcal{E}$, i.e., a feasible solution to the ESP, is $\mathcal{NP}$-complete in the strong sense.

Two Phase Method: Addition of Artificial Variables

A simple strategy would be starting with $\Omega' = \emptyset$, which is conducted similar in spirit to the first phase of the simplex method, and has been proposed e.g., by DUMAS, DESROSIERS & SOUMIS (1991) and SOL (1994). Consider the modified restricted master program, analogous to (2.6),

$$\begin{array}{rlrcll} \min & \displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega'_e} c_R^e \lambda_R^e & + & \displaystyle\sum_{r\in\mathcal{R}} M y_r & & \\ \text{subject to} & \displaystyle\sum_{e\in\mathcal{E}}\sum_{R\in\Omega'_e} \delta_{rR} \lambda_R^e & + & y_r & = & 1, \quad r \in \mathcal{R} \\ & \displaystyle\sum_{R\in\Omega'_e} \lambda_R^e & & & \leq & 1, \quad e \in \mathcal{E} \\ & \lambda_R^e & & & \geq & 0, \quad e \in \mathcal{E},\ R \in \Omega'_e \\ & & & y_r & \geq & 0, \quad r \in \mathcal{R} \ , \end{array} \tag{4.2}$$

where $M > \max_{e\in\mathcal{E}, R\in\Omega_e} c_R^e$ is a penalty cost, chosen in such a way that the incorporation of artificial variables y_r, $r \in \mathcal{R}$, into a primal solution is suppressed. Clearly, the unit columns corresponding to these variables constitute a basis matrix feasible for the modified formulation. As soon as all artificial variables become non-basic during the column generation process, a feasible solution to the original (RMP′)—if existent—is found. It was pointed out by SOL (1994) that it makes sense to keep the non-basic artificial variables in the formulation. In a branch-and-price approach, branching may result in an initially infeasible restricted master program in a

node, and the first phase is automatically resumed. We will see in Subsection 4.2.5 that some care has to be taken when choosing a value for M.

Heuristics

An alternative (or a complement) to this *all artificial* start is the heuristic construction of feasible tours. Even if we fail to incorporate *all* requests in the initial solution—which is likely because of the discouraging complexity status of the feasibility problem—we are better off with the total cost. Classical nearest neighbor strategies appear not to work very well (DESSARPS 2000). We describe some simple different approaches.

Greedy Assignment

Among a bunch of possibilities, two straight forward *greedy* construction strategies incrementally append patterns to initially empty concatenations. Their pseudo-codes read as follows.

```
// Greedy Patterns
for all e ∈ E do
   concat(e) ← {e+}
end for
remain ← P

repeat
   choose best P̄ ∈ remain
   choose best ē ∈ E for P̄
   if concat(ē) ∪ {P} is feasible then
      append P at concat(ē)
   end if
   remain ← remain \ {P̄}

until remain = ∅
```

```
// Greedy Engines
for all e ∈ E do
   concat(e) ← {e+}
end for
remain ← P
ē ← 1
repeat

   choose best P̄ ∈ remain for ē
   if concat(ē) ∪ {P̄} is feasible then
      append P at concat(ē)
   end if
   remain ← remain \ {P̄}
   ē ← ē + 1  mod |E|
until remain = ∅
```

In the *greedy engines* heuristic the engines cyclically choose in turn the next pattern to be served on their respective concatenation, minimizing e.g., the engine's travel time. This heuristic tends to produce solutions where the length of concatenations is balanced between the engines. In the *greedy patterns* heuristic, patterns are assigned to a best possible engine on which they are to be served, possibly according to the same criterion as before. However, in a first stage, local criteria determine the order in which the patterns are considered until no pattern is left to be feasibly assigned to an engine. A standard order of patterns P is chronologically, e.g., in non-decreasing order of $\underline{t}_{o(P)}$. Another choice is the *minimal remaining time slack* criterion according to which, in a sense, the most time critical pattern P is considered first, i.e., one with minimal $\sum_{i \in P}(\overline{t}_i - \underline{t}_i)$. It has the smallest *degree of freedom* with respect to floating within its

time windows and should be scheduled as early as possible in order to prevent infeasibilities. A third mode of assignment is simply at random.

Improvement by Arc Exchanges

The concatenations constructed by simple heuristics as those just presented are likely to have a poor quality in terms of cost. Worse than that, some requests are possibly not assigned to any engine at all. This results in an infeasible initial solution to the ESP, and artificial variables have to be introduced in the set partitioning formulation as well.

A generic improvement procedure for routing problems relies on the concept of a k-exchange: k arcs present in a solution are replaced by k other arcs, such that the result is feasible, and the objective function value does not deteriorate. The two arc sets are not necessarily required to be disjoint. The exchange is repeated until no further improvement is possible. Since the complexity exponentially increases in k, a common choice is $k \in \{2,3\}$. It is more sophisticated to individually determine k for each exchange. As long as there are additional cost savings realizable for $k+1$ in a particular iteration, k is increased. Such a *variable-depth search* is proposed by VAN DER BRUGGEN, LENSTRA & SCHUUR (1993) for the 1-PDPTW. The method is extendable to the multiple vehicle case, as discussed by KINDERVATER & SAVELSBERGH (1997). These authors also report on efficient implementation strategies to preserve feasibility of a solution in presence of e.g., precedence or time window constraints.

In order to preserve its special structure when applying k-exchanges to a concatenation C, it is indicated to distinguish between *inter* and *intra* pattern arcs. Whereas the latter are given by $\dot{\bigcup}_{P \subseteq C} \mathcal{A}|_P$, all arcs connecting two patterns constitute the former. Naturally, only inter pattern arcs should be subject to arc exchange procedures described above. We introduce two additional operations to be performed (also) on intra pattern arcs, c.f. Figure 4.1. *Splitting* of a pattern in $\mathcal{P}^2$

Figure 4.1: Example of splitting and blending of two requests

results in two consecutively visited full truckload patterns. Among the two possible sequences we choose one with smallest cost. *Blending* is the inverse operation, and collapses two consecutive full truckload patterns into an overlapping or an embedding pattern. Among the four potential outcomes one with smallest cost is chosen. In order to avoid cycling by splitting and blending the same two requests back and forth, only strictly cost improving moves are eligible.

We finally address the problem of time infeasible concatenations. In the above construction heuristics requests remain unassigned because of violated time windows. When we append another yet unvisited pattern to an improved feasible concatenation, still infeasibilities may be present, but will be tentatively tolerated. We measure *total infeasibility* of a concatenation C by

$\sum_{i\in C}\max\{0, T_i - \bar{t}_i\}$, where T_i denotes the start-of-service time in location i. The above improvement operations are admissible if at least one of cost or total infeasibility of the concatenation is strictly decreased. The this way modified two phases splitting/blending and arc exchange are alternately executed until neither improvement takes places. If the eventually obtained concatenation is feasible, we iterate. The longest constructed feasible concatenation is returned.

Backtracking

Ideally, given the set R of requests visited by a particular engine, we would like to improve the corresponding concatenation as much as possible. Such a procedure is an additional or alternative option to the above, and a simple backtracking procedure is capable of achieving this goal. A concatenation is recursively extended by feasibly appending a pattern of least cost at the end, c.f. Algorithm 4.1. At each stage of the recursion we are given the set $S \subseteq R$ of already scheduled requests, the last visited location i in the associated concatenation, the arrival time T_i in i, and the corresponding cost C_i. For engine $e \in \mathcal{E}$ the initial call is with $S = \emptyset$, $i = e^+$, $T_i = \underline{t}_{e^+}$, and $C_i = 0$.

Algorithm 4.1 Calculate Optimal Cost of a $\mathcal{P}^{1\cup 2}$-Concatenation for a Given Request Set $R \subseteq \mathcal{R}$

parameter: S, i, T_i, C_i
if $S = R$ **then**
  return 0
end if
$C_{best} \leftarrow \infty$
for all $P \in \mathcal{P}$ with requests$(P) \subseteq R$ **and** requests$(P) \cap S = \emptyset$ **do**
  $j_1 \leftarrow i$
  $\tilde{T} \leftarrow T_i$
  $\tilde{C} \leftarrow C_i$
  for $j_2 = o(P), \ldots, d(P)$ **do** // consider all nodes in P in order
    if $\tilde{T} + s_{j_1} + t_{j_1 j_2} \leq \bar{t}_{j_2}$ **then**
      $\tilde{T} \leftarrow \max\{\underline{t}_{j_2},\ \tilde{T} + s_{j_1} + t_{j_1 j_2}\}$
      $\tilde{C} \leftarrow \tilde{C} + \bar{c}_{j_1 j_2}$
    else
      Expansion is time infeasible; proceed to next P
    end if
    $j_1 \leftarrow j_2$
  end for
  $\tilde{C} \leftarrow \tilde{C} +$ Algorithm 4.1$(S \setminus$ requests$(P), d(P), \tilde{T}, \tilde{C})$
  if $\tilde{C} < C_{best}$ **then**
    $C_{best} \leftarrow \tilde{C}$ // Then, in particular, $C_{best} < \infty$
  end if
end for
return C_{best}

When an integer solution to the restricted master program is found, a modified version of Algorithm 4.1 is used to reconstruct also the concatenations from their encodings as column incidence vectors.

4.1.2 Heuristics at the Master Level

Decomposition of the Planning Horizon For large instances of the ESP a reasonably quick heuristic solution can be computed by decomposing the planning horizon into overlapping time slices. The set of requests is chronologically sorted, e.g., by non-decreasing $\underline{t}_{r^+}$, $r \in \mathcal{R}$. Then, instances of manageable size are successively solved as follows.

Algorithm 4.2 Temporal Decomposition of the ESP

repeat
  choose first k requests $\mathcal{R}^1$ from chronologically sorted $\mathcal{R}$
  solve ESP for $\mathcal{R}^1$; engine $e \in \mathcal{E}$ finishes in e^- at time T_{e^-}
  $t \leftarrow t - \min_{e\in\mathcal{E}} T_{e^-} \quad \forall t \in \bigcup_{i\in\mathcal{N}}\{\underline{t}_i, \bar{t}_i\}$ // "translate time"
  $e^+ \leftarrow e^- \quad \forall e \in \mathcal{E}$ // "move engines"
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \mathcal{R}^1$
until $\mathcal{R} = \emptyset$

Temporarily Fixed Variables In the event that a variable λ_R^e attains a value close to one, say greater than 0.99, this variable is fixed for some (possibly even all remaining) iterations of the column generation process. As a consequence the pricing problem may become easier to solve. The number of iterations is either preset, or dynamically determined by observing the dual variable values u_r, $r \in R$. As soon as the temporarily disregarded requests become attractive from a reduced cost point of view, λ_R^e is released again.

4.2 Pricing Problem

The pricing problem (4.1) can be decomposed into subproblems, *viz.* one for each $e \in \mathcal{E}$,

$$z_e^\star := \min\left\{ c_R^e - \sum_{r\in R} u_r - v_e \mid R \in \Omega_e \right\}, \tag{4.3}$$

where the dual variable value v_e corresponding to engine $e \in \mathcal{E}$ constitutes an additive constant not interfering with the minimization process. The minimum $\min_{e\in\mathcal{E}} z_e^\star$ determines a column to be adjoined to the restricted master program. We call (4.3) the *engine scheduling pricing problem*, or ESPP for short. It consists of finding a shortest (request disjoint) $\mathcal{P}^{1\cup 2}$-concatenation for one fixed engine with the additional cost of $-u_r \in \mathbb{R}$ incurred for each visited request $r \in \mathcal{R}$. We will

occasionally use the notion of looking for an *ESPP-concatenation*. Note, that in this problem we are not required to visit *all* the requests, but only a (possibly empty) subset. Thence, we first need to clarify its computational complexity status.

4.2.1 Complexity of the Pricing Problem

For the purpose of this subsection we restate ESPP as follows. Given a request graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with arc weights $c_{ij} \in \mathbb{R}_+$, $(i,j) \in \mathcal{A}$ and node weights $u_r \in \mathbb{R}$, $r^- \in \mathcal{N}$, and zero otherwise, and an integer K. Is there an ESPP-concatenation of length K or less?

Proposition 4.3 *ESPP is $\mathcal{NP}$-complete in the strong sense.*

Proof. Constructing an arbitrary ESPP-concatenation in $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ and evaluating its length is polynomial in $|\mathcal{R}|$, thus ESPP is in $\mathcal{NP}$.

Completeness in the strong sense is shown by reduction from (directed) LONGEST PATH (GAREY & JOHNSON 1979, problem ND29), an instance of which is given by a directed graph $G_{\mathsf{LP}} = (V_{\mathsf{LP}}, A_{\mathsf{LP}})$, arc weights $w_{ij} \in \mathbb{Z}_+$ for $(i,j) \in A_{\mathsf{LP}}$, two specified nodes $s, t \in V_{\mathsf{LP}}$, and an integer K. The question is whether there exists a node disjoint directed path in G_{LP} from s to t of length K or more.

Consider the following (polynomial) transformation into an ESPP instance, c.f. Figure 4.2 on the next page. At first, introduce two distinct nodes e^+, e^-, and split each $i \in V_{\mathsf{LP}}$ in two nodes i^+, i^-. This yields the node set $\mathcal{N}$ of the request graph. The arc set $\mathcal{A}$ is defined as usual, c.f. Equation (1.1) on page 10. Let $u_i := -\max\{w_{ij} \mid (i,j) \in A_{\mathsf{LP}}\}$ for all $i \in N_{\mathsf{LP}}$. The arc weights are defined as follows:

$$
\begin{array}{rcrcll}
c_{e^+s^+} & = & c_{t^-e^-} & = & 0 & \\
 & & c_{i^+i^-} & = & 0 & \text{for all } i \in V_{\mathsf{LP}} \\
 & & c_{i^-j^+} & = & -w_{ij} - u_i & \text{for all } (i,j) \in A_{\mathsf{LP}} \\
 & & c_{ij} & = & M & \text{otherwise,}
\end{array}
$$

where M is a sufficiently large number, e.g., $M = \sum_{(i,j)\in A_{\mathsf{LP}}} w_{ij} + 1$. Let the service time be zero for all locations. All time windows are made irrelevant by widening them to the whole planning horizon. We allow $\mathcal{P}^1$-concatenations only, e.g., by setting $L_e = 1$ and $\ell_i = 1$, $i \in N_{\mathsf{LP}}$, which in fact is redundant because no arc $(i^+, j^+) \in \mathcal{A}$ will be part of an optimal concatenation because of its large weight. By contraction of the split nodes, an optimal solution P_{ESPP} of length K to the above ESPP instance corresponds one-to-one to an optimal solution P_{LP} to the original LONGEST PATH instance of length $-K$, since

$$
\sum_{(i,j)\in P_{\mathsf{LP}}} w_{ij} = \sum_{(i,j)\in P_{\mathsf{LP}}} -u_i - c_{i^-j^+} = -\left(\sum_{\delta(i^-)\in P_{\mathsf{ESPP}}} u_i + \sum_{(i,j)\in P_{\mathsf{ESPP}}} c_{ij} \right) .
$$

This completes the proof. □

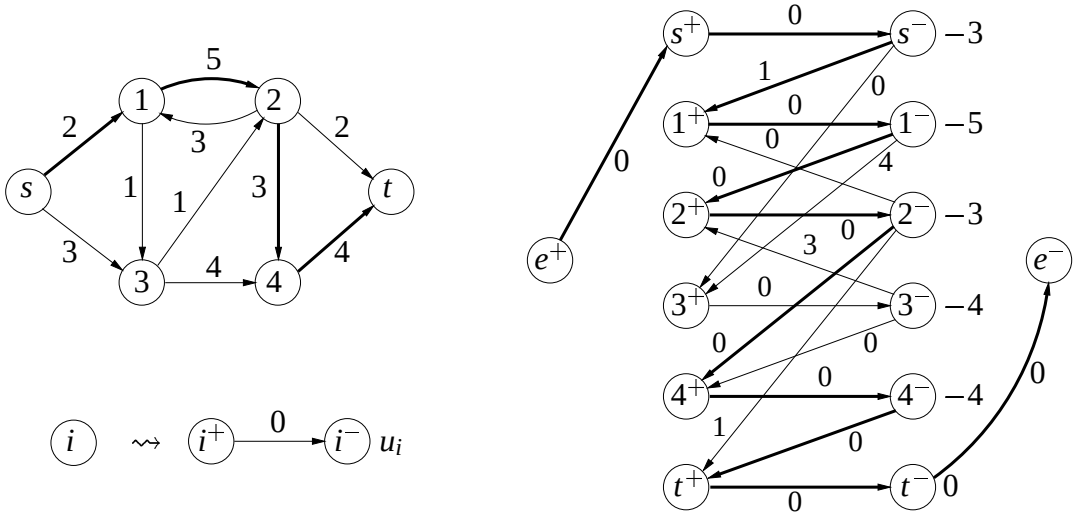

Figure 4.2: Construction of an ESPP instance from a LONGEST PATH instance

From the proof's transformation it is clear that restricting attention to $\mathcal{P}^1$-concatenations does not change the computational complexity of the ESPP. Interestingly, the problem remains difficult even in absence of time windows.

In the remainder of this section we develop exact and heuristic algorithms for the ESPP. Probably the first exact approach which comes to mind is the formulation as a mixed integer program stemming from (ESPMIP). Indeed, Table 4.1 on the facing page presents such a model adapted to a single engine, using the same notation—and interpretation—as in Table 3.1 of the preceding chapter. The objective function (4.4) is now to minimize the *reduced cost*. Note, that (4.5) is a *packing constraint* and replaces the requirement for a (trivial) partition of requests. Since this model suffers from basically the same weaknesses as (ESPMIP) itself, c.f. Section 3.1, we are interested in a more tailored algorithm.

4.2.2 Constrained Shortest Path Problems

The ESPP as well as the 1-ESP are shortest path problems with additional (linear) constraints. This family of problems has received considerable attention, and will be considered here in order to familiarize ourselves with some generic algorithmic ideas employed for its solution.

Given a directed graph $G = (N,A)$ with node set N and arc set A, two distinguished nodes $e^+, e^- \in N$, arc lengths $c_{ij} \in \mathbb{R}$, $(i,j) \in A$, and a *resource consumption* $t_{ij} \in \mathbb{R}_+$, $(i,j) \in A$ along each arc. The *resource constrained shortest path problem* (*see* MEHLHORN & ZIEGELMANN (2000), and the references therein) asks for the construction of a directed[1] path P from e^+ to e^- which minimizes $\sum_{(i,j)\in P} c_{ij}$ such that the total resource consumption along P is within given bounds, i.e., $\underline{t} \leq \sum_{(i,j)\in P} t_{ij} \leq \bar{t}$. This problem is $\mathcal{NP}$-complete (GAREY & JOHNSON 1979).

[1] Since only the directed versions of the problems are of relevance here, we henceforth drop the attribute *directed*.

$$\text{minimize} \quad T_{e^-} - \sum_{P \in \mathcal{P}_e} z_P \cdot \sum_{r^+ \in P} u_r - v_e \tag{4.4}$$

$$\text{subject to} \quad \sum_{P \in \mathcal{P}_e} \delta_{rP} z_P \leq 1 \qquad \forall r \in \mathcal{R} \tag{4.5}$$

$$\sum_{P \neq P_1 \in \mathcal{P}_e \cup \{e^+\}} x_{P_1 P} = \sum_{P \neq P_2 \in \mathcal{P}_e \cup \{e^-\}} x_{P P_2} = z_P \qquad P \in \mathcal{P}_e \tag{4.6}$$

$$\sum_{P \in \mathcal{P}_e \cup \{e^-\}} x_{\{e^+\} P} = \sum_{P \in \mathcal{P}_e \cup \{e^+\}} x_{P \{e^-\}} = 1 \tag{4.7}$$

$$z_P = 1 \;\Rightarrow\; T_i + s_i + t_{ij} \leq T_j \qquad P \in \mathcal{P}_e, (i,j) \in \mathcal{A}|_P \tag{4.8}$$

$$x_{P_1 P_2} = 1 \;\Rightarrow\; T_{d(P_1)} + s_{d(P_1)} + t_{d(P_1) o(P_2)} \leq T_{o(P_2)} \qquad P_1 \neq P_2 \in \mathcal{P}_e \cup \{e^+, e^-\} \tag{4.9}$$

$$\underline{t}_i \leq T_i \leq \bar{t}_i \qquad \forall i \in \mathcal{N} \tag{4.10}$$

$$z_P \in \{0,1\} \qquad P \in \mathcal{P}_e \tag{4.11}$$

$$x_{P_1 P_2} \in \{0,1\} \qquad P_1 \neq P_2 \in \mathcal{P}_e \cup \{e^+, e^-\} \tag{4.12}$$

Table 4.1: Mixed integer formulation for the engine scheduling pricing problem

A generalization introduced by DESROSIERS, PELLETIER & SOUMIS (1983) is the *shortest path problem with time windows* (SPPTW). Here, time windows $[\underline{t}_i, \bar{t}_i]$ are given for each $i \in N$, within which the node must be visited in a shortest path solution. Time is the resource in this setting, its consumption $t_{ij} \in \mathbb{R}_+$ is given on arcs $(i,j) \in A$, but usually, time is *not* merely additive along paths. Instead, *waiting* is allowed before a time window opens. The multi-dimensional generalization of the SPPTW with K resources is called the *shortest path problem with resource windows*,[2] where resource consumption $t^k_{ij} \in \mathbb{R}_+$, $(i,j) \in A$, and resource windows $[\underline{t}^k_i, \bar{t}^k_i]$, $i \in N$, are given for each $k = 1, \ldots, K$ (*see* e.g., DESROSIERS, DUMAS, SOLOMON & SOUMIS 1995).

Dynamic Programming

All the above problems can be solved by dynamic programming as we will exemplify for the SPPTW, c.f. DESROCHERS, DESROSIERS & SOLOMON (1992). Recall, that the formal description of a dynamic programming algorithm requires the introduction of an appropriate *state space representation* of (partial) solutions. Furthermore, *state transitions* need to be specified by means of *recurrence formulae*. All states that correspond to feasible solutions of the original problem are called *final states*. IBARAKI (1987) exhaustively discusses all of these notions.

Denote by $C(R,i,T_i)$ the minimal cost of a (time window feasible) node disjoint path from e^+ to $i \in N$, visiting exactly $R \subseteq N$ with arrival time T_i in node i. The following recurrence formulae

[2] This problem is also referred to as the *resource constrained shortest path problem*, especially in the large body of literature available from the Montréal research groups. Here, we use a different terminology in order to avoid the naming conflict.

describe the computation of the values $C(R,i,T_i)$:

$$C(\emptyset,e^+,\underline{t}_{e^+}) = 0 \tag{4.13}$$

$$C(R,j,T_j) = \min_{(i,j)\in A} \left\{ C(R\setminus\{j\},i,T_i) + c_{ij} \mid i \in R\setminus\{j\},\ T_i + t_{ij} \le T_j,\ \underline{t}_i \le T_i \le \bar{t}_i \right\} \tag{4.14}$$

for all $R \subseteq N$, $j \in N$, and $\underline{t}_j \le T_j \le \bar{t}_j$. Moreover, we define $C(R,j,T_j) = C(R,j,\underline{t}_j)$ for $T_j < \underline{t}_j$, and $C(R,j,T_j) = \infty$ for $T_j > \bar{t}_j$. Then, inspection of the final states

$$\min_{R\subseteq N} \min_{\underline{t}_{e^-} \le T_{e^-} \le \bar{t}_{e^-}} C(R,e^-,T_{e^-}) \tag{4.15}$$

yields an optimal solution to the SPPTW. We identify the classical *principle of optimality* on which Equations (4.14) rely: When the minimum at a certain stage of the algorithm is attained for $C(R,j,T_j)$, and arc (i,j) is last in the corresponding solution, then $C(R\setminus\{j\},i,T_i)$ is minimal at the preceding stage. Note, that for the state space to be finite it is assumed that only discrete, usually integral, arrival times $\underline{t}_i \le T_i \le \bar{t}_i$ will occur—which is certainly no loss of generality for practical problems.

The node disjoint SPPTW is $\mathcal{NP}$-complete in the strong sense (DROR 1994), i.e., not even pseudo-polynomial algorithms exist. For its use in the column generation context it is legitimate to consider a relaxation for which pseudo-polynomial algorithms have been proposed. To this end, we waive the node disjoint path requirement, i.e., multiple visits to nodes are allowed. Consequently, the set R of visited nodes needs no longer be maintained for each state, and Equations (4.13) – (4.15) are simplified accordingly (DESROCHERS, DESROSIERS & SOLOMON 1992). To make use of this possibility, an *overcovering* of requests must be allowed, i.e., the partitioning constraints in (RMP′) need to be relaxed to their covering analogues $\sum_{e\in\mathcal{E}} \sum_{R\in\Omega'_e} \delta_{rR}\lambda^e_R \ge 1$, $r \in \mathcal{R}$. In fact, research has been focusing on pseudo-polynomial algorithms for solving relaxations of the strongly $\mathcal{NP}$-complete node disjoint problem version.

Parenthesis: Shortest Path Algorithm for ESPP on a Time Expanded Graph

The SPPTW can be solved by application of a shortest path algorithm to a time expanded graph, i.e., each node $i \in N$ is duplicated $\bar{t}_i - \underline{t}_i + 1$ times, once for each feasible moment to be visited. Arcs exist between nodes if and only if traveling between them is time feasible. When $t_{ij} > 0$, $(i,j) \in A$, which is a usual assumption in the literature, the resulting graph is *acyclic*, rendering shortest path algorithms particularly simple. This proceeding, in a similar way, is also an option for the construction of *non request disjoint* $\mathcal{P}$-concatenations as well as non request disjoint ESPP-concatenations, when the objective is to minimize the total travel time. For its presentation and further development we introduce a new structure, illustrated in Figure 4.3 on page 98.

Definition 4.4 (Pattern Graph)
Given a family $\mathcal{P}$ of patterns, the associated pattern graph $\mathcal{G}_\mathcal{P} = (\mathcal{P}, \mathcal{A}_\mathcal{P})$ is defined by the node set $\mathcal{P}$ and the arc set $\mathcal{A}_\mathcal{P}$ given by $(P_1,P_2) \in \mathcal{A}_\mathcal{P}$ if and only if

(i) $P_1 \cap P_2 = \emptyset$

(ii) $\underline{t}_{d(P_1)} + s_{d(P_1)} + t_{d(P_1)o(P_2)} \leq \bar{t}_{o(P_2)}$.

We can represent request disjoint $\mathcal{P}$-concatenations as simple paths in a pattern graph, but the converse is not true since repetitions of requests may occur and time windows may be violated. The definition of the arc set eliminates such incompatibilities at least between successive nodes. We give the design of the time expanded pattern graph, c.f. Figure 4.4 on the following page, in Algorithm 4.6 on page 99 (we assume a single origin and a single destination), which is an adapted shortest path algorithm for solving the non request disjoint ESPP-concatenation problem. That is, avoiding multiple visits to requests is not taken care of by this kind of algorithm. The computational complexity status of the relaxed problem, however, is improved as compared to its request disjoint counterpart.

Lemma 4.5 *Algorithm 4.6 for solving the non request disjoint ESPP-concatenation problem has pseudo-polynomial running time* $O(|\mathcal{R}|^4 \cdot T^2)$*, where* $T = \max_{i \in \mathcal{N}}(\bar{t}_i - \underline{t}_i + 1)$.

Proof. With $|\mathcal{P}^{1\cup 2}|$ being quadratic in $|\mathcal{R}|$, and $|P| \leq 4$ for $P \in \mathcal{P}^{1\cup 2}$, the number of nodes in the time expanded pattern graph is $\sum_{P\in\mathcal{P}} \sum_{i\in P}(\bar{t}_i - \underline{t}_i + 1) + 2 = O(|\mathcal{R}|^2 \cdot T)$. Although being far from complete, the graph's number of arcs is quadratic in the number of nodes, i.e., $O(|\mathcal{R}|^4 \cdot T^2)$. The topological order of the nodes needed for the algorithm can be computed in time linear in the number of arcs. The claim now follows from the fact, that a shortest path in an acyclic graph can be found using the reaching algorithm in time linear in the number of arcs as well (*see* AHUJA, MAGNANTI & ORLIN 1993, Theorem 4.3). □

We remark, that in the ESPP context, a possible repetition of visited requests does not only originate from the algorithmic design but also from the inevitable duplication of requests occurring in multiple nodes of the pattern graph. Another severe drawback is that even for small instances of the ESPP with wider time windows, the number of nodes (and arcs) of the time expanded pattern graph may become exorbitant. An algorithm which works directly on the pattern graph or even on the request graph would be more desirable. For its development we switch back to the SPPTW again for the remainder of this subsection.

Considering Time Window Constraints in Labeling Algorithms

An important concept in the design of combinatorial shortest path algorithms is to maintain a set of *distance labels*, one for each node $i \in N$ of the graph. Starting at infinity and being improved in the iterative course of the algorithms, its value is at any time an upper bound on the length of a shortest path from the source e^+ to node i, equaling the respective optimal length upon termination. We have seen this concept at work already in Algorithm 4.6. When we wish to keep track of both, time *and* cost, however, *one-dimensional* labels do not suffice. DESROSIERS, PELLETIER & SOUMIS (1983) therefore introduce for the SPPTW for a time window feasible

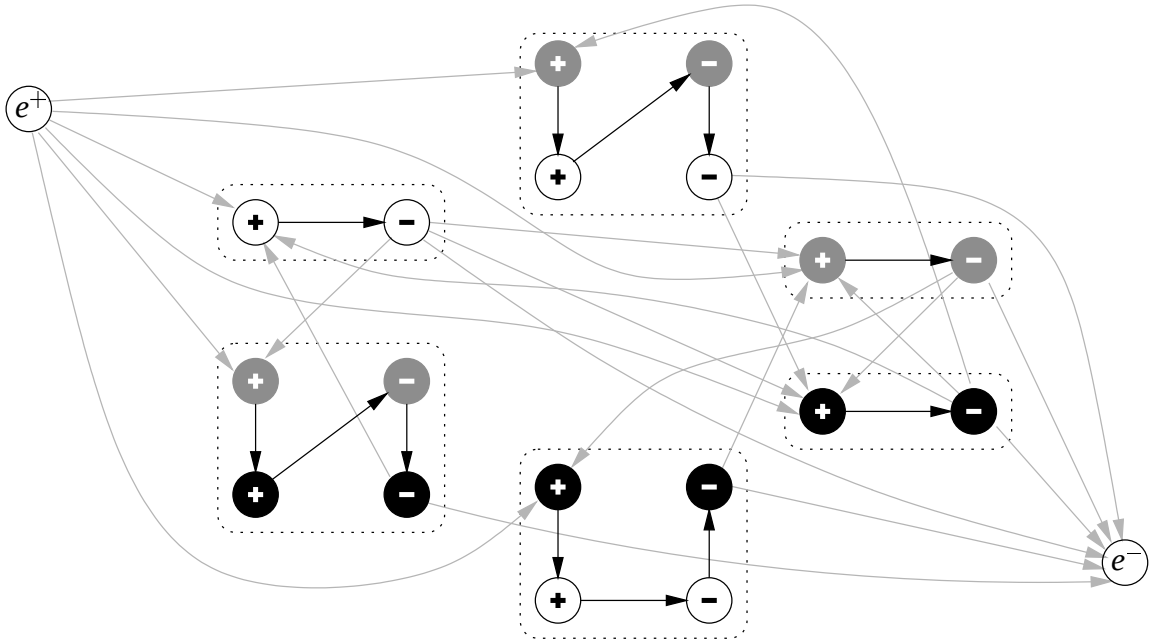

Figure 4.3: An example pattern graph involving three requests (symbolized by ○, ●, and ●)

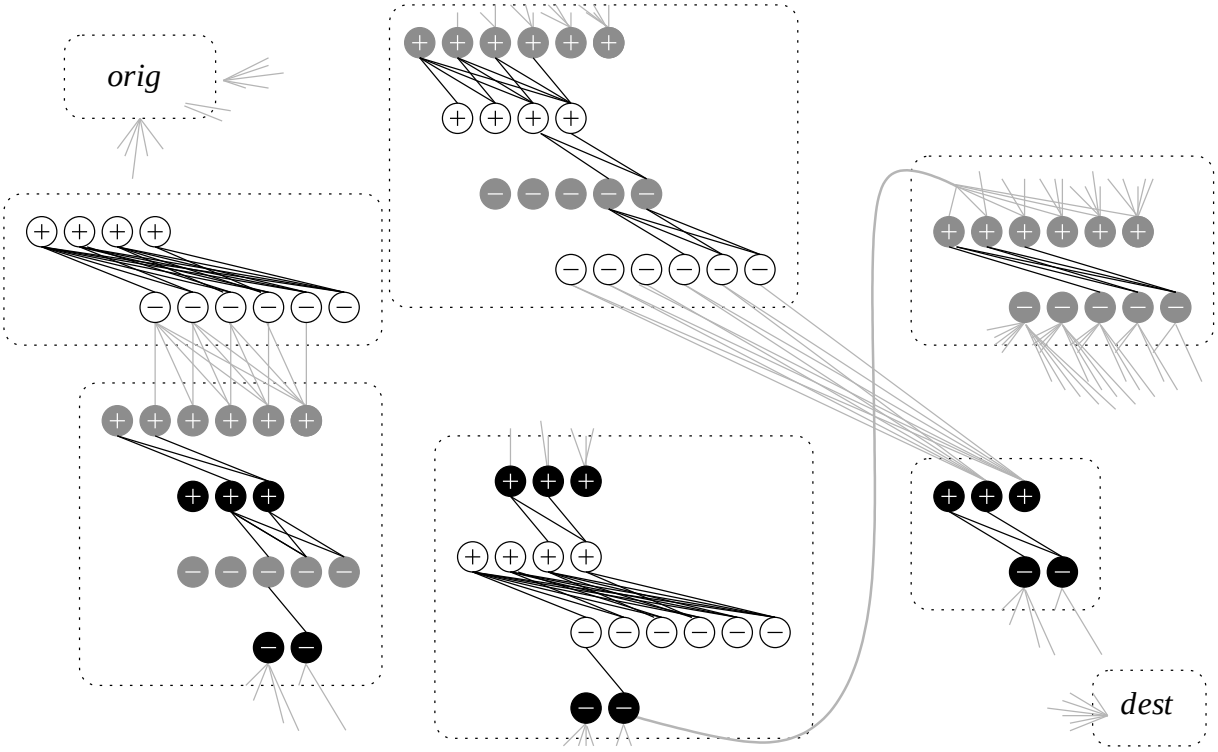

Figure 4.4: Schematic excerpt from a time expanded pattern graph corresponding to Figure 4.3

Algorithm 4.6 Solving ESPP on a Time Expanded Pattern Graph $\mathcal{G}_{\mathcal{P}}^{\text{time}} = (\mathcal{N}_{\mathcal{P}}^{\text{time}}, \mathcal{A}_{\mathcal{P}}^{\text{time}})$

// Construction of the time expanded pattern graph
$\mathcal{N}_{\mathcal{P}}^{\text{time}} \leftarrow \bigcup_{P\in\mathcal{P}} \bigcup_{i\in P} \bigcup_{t\in\{\underline{t}_i,\ldots,\bar{t}_i\}} (P,i,t) \cup \{orig := (\{e^+\}, e^+, \underline{t}_{e^+}),\ dest := (\{e^-\}, e^-, \bar{t}_{e^-})\}$
$\mathcal{A}_{\mathcal{P}}^{\text{time}} \leftarrow \emptyset$
for all $P \neq Q \in \mathcal{P} \cup \{e^+, e^-\}$ **do**
  for all $(i,j) \in \mathcal{A}|_P$ **do**
    for all $(P,i,a) \neq (P,j,b) \in \mathcal{N}_{\mathcal{P}}^{\text{time}}$ **do**
      if $a + s_i + t_{ij} \leq b$ **then**
        $\mathcal{A}_{\mathcal{P}}^{\text{time}} \leftarrow \mathcal{A}_{\mathcal{P}}^{\text{time}} \cup ((P,i,a),(P,j,b))$ // edges within a pattern
      end if
    end for
  end for
  for all $(P,d(P),a) \neq (Q,o(Q),b) \in \mathcal{N}_{\mathcal{P}}^{\text{time}}$ **do**
    if $a + s_{d(P)} + t_{d(P)o(Q)} \leq b$ **then**
      $\mathcal{A}_{\mathcal{P}}^{\text{time}} \leftarrow \mathcal{A}_{\mathcal{P}}^{\text{time}} \cup ((P,d(P),a),(Q,o(Q),b))$ // edges connecting patterns
    end if
  end for
end for
// Adapted reaching algorithm
for all $r \in \mathcal{R}$ **do**
  $u(r^+) \leftarrow u_r$ // dual variable value corresponding to $r \in \mathcal{R}$
  $u(r^-) \leftarrow 0$
end for
$d(orig) \leftarrow 0$
for all $(P,i,t) \in \mathcal{N}_{\mathcal{P}}^{\text{time}} \setminus \{orig\}$ **do**
  $d((P,i,t)) \leftarrow \infty$
end for
for all $(P,i,a) \in \mathcal{N}_{\mathcal{P}}^{\text{time}}$ considered in ascending topological order **do**
  for all $((P,i,a),(Q,j,b)) \in \delta((P,i,a))$ **do** // arcs emanating from (P,i,a)
    if $d(P,i,a) + \max\{a + s_i + t_{ij},\ b\} - u(j) < d(Q,j,b)$ **then**
      $d(Q,j,b) \leftarrow d(P,i,a) + \max\{a + s_i + t_{ij},\ b\} - u(j)$
    end if
  end for
end for
// Shortest ESPP-concatenation has weight, i.e., reduced cost $d(dest)$

path from the source to node i a label (T_i^k, C_i^k), representing the start-of-service time T_i^k at node i, and the corresponding cost C_i^k of the k^{th} path from the source to node i. We occasionally drop the superscript k. Actually, when waiting incurs no cost, there may be many paths with cost C_i, arriving at node i at time T_i or earlier. We therefore identify the existence of label (T_i, C_i) with the existence of *some* path as defined above.

Definition 4.7 (Label Dominance)
Label (T_i^1, C_i^1) is said to dominate label $(T_i^2, C_i^2) \neq (T_i^1, C_i^1)$ if and only if $T_i^1 \leq T_i^2$ and $C_i^1 \leq C_i^2$.

Definition 4.8 (Efficient Label)
A label (T_i, C_i) at node i is said to be efficient if and only if no other labels at node i dominate it.

Contrasting the ordinary shortest path problem, the label dominance relation does not define a total, but a *partial order* of labels. Hence, we are to maintain at least the set of efficient labels at each node. In fact, it can be inductively proven that the efficient labels suffice. Accounting for this fact, the classical label correcting algorithm by BELLMAN and FORD is adapted in Algorithm 4.9 on the facing page (DESROSIERS, PELLETIER & SOUMIS 1983). The fundamental step is the *treatment* of a label (T_i^k, C_i^k), i.e., the construction of all feasible extensions of the corresponding path, each by one additional arc $(i, j) \in \delta(i)$, resulting in new paths and the associated labels, respectively. Treating all the labels at a node is called the treatment of this node. Each time, new labels are generated at node i, these may dominate others already present at node i, thus offering possible improvement. Then, node i must be treated (again) in order to propagate this improvement. The process terminates as soon as no new labels are generated at any node. Algorithm 4.9 keeps a list $\mathcal{T}$ of nodes still to be treated, and lists $\mathcal{L}_i$, $i \in N$, containing all labels present at node i, respectively.

The algorithm is ambiguous about the *order of treatment* of nodes, which in turn determines the number of times each node is treated. Several variants have been proposed like FIFO, LIFO, and various queueing techniques. An optimal ordering would ensure each node to be treated exactly once, which actually would be true for a topological order of nodes if the underlying graph was acyclic. DESROSIERS, PELLETIER & SOUMIS (1983) report on a substantial reduction in terms of computation time when nodes are ordered chronologically, e.g., according to non-decreasing $\underline{t}_i$, $i \in N$.

Remark. The idea of *forward* dynamic programming, which is inherently present also in the labeling algorithms, has an important benefit. We can handle much more complicated objective functions than linear ones, e.g., cost depending on the status of the vehicle, i.e., loaded or empty.

Label Management

When it comes to implementing the aforementioned or next to be presented labeling algorithms, at least two issues regarding the labels need to be settled: In what order to treat labels, and how to sort out dominated labels efficiently?

Algorithm 4.9 Label Correcting Algorithm for SPPTW

// Initialization
$\mathcal{L}_{e^+} \leftarrow \{(T^1_{e^+} := \underline{t}_{e^+},\, C^1_{e^+} := 0)\}$
for all $i \in N \setminus \{e^+\}$ **do**
  $\mathcal{L}_i \leftarrow \{(T^1_i := \underline{t}_i,\, C^1_i := \infty)\}$
end for
$\mathcal{T} \leftarrow \{e^+\}$
// Treatment of node i
while $\mathcal{T} \neq \emptyset$ **do**
  Choose a node $i \in \mathcal{T}$
  for all $(i,j) \in \delta(i)$ **do**
    for all $(T^k_i, C^k_i) \in \mathcal{L}_i$ **do**
      if $T^k_i + t_{ij} \leq \overline{t}_j$ **then**
        $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{(\max\{\underline{t}_j,\, T^k_i + t_{ij}\},\, C^k_i + c_{ij})\}$
      end if
    end for
    Delete dominated labels from $\mathcal{L}_j$
    if $\mathcal{L}_j$ was changed by treating node i **then**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$
    end if
  end for
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \{i\}$
end while
// Each least cost label in $\mathcal{L}_{e^-}$ determines a shortest path with time windows

Firstly, observe that for a set of efficient labels ordered by strictly[3] increasing time we have

$$T^1_i < T^2_i < \cdots < T^k_i \quad \Longrightarrow \quad C^1_i > C^2_i > \cdots > C^k_i \ , \tag{4.16}$$

which holds because all labels (T_i, C_i) with $T_i \in [T^\kappa_i, T^{\kappa+1}_i)$ and $C^\kappa_i \leq C_i$, $\kappa \in \{1,\ldots,k-1\}$ are dominated by (T^κ_i, C^κ_i). Therefore, represented as time intervals in the T-C plane, efficient labels constitute a decreasing staircase function, c.f. part (b) of Figure 4.5 on the next page. Having the list of labels for each node sorted this way we can eliminate dominated labels in linear time by a sequential inspection.

Practically, this elimination may still be an expensive operation for large lists of labels, and we should prevent efforts from growing unduly. In fact, at each node i we only need to treat labels modified or inserted in $\mathcal{L}_i$ no sooner than the iteration in which node i was last inserted in $\mathcal{T}$. The labels in this sublist $\mathcal{L}'_i \subseteq \mathcal{L}_i$ are called *recent*. Consider Figure 4.5 (a) for an illustration. This improvement was already suggested by DESROSIERS, PELLETIER & SOUMIS (1983).

The treatment of labels in the manner so far described is called *reaching* in allusion to one fixed node from which the respective paths are extended. New labels are thus generated at dif-

[3] If $T^\kappa_i = T^{\kappa+1}_i$ for some $\kappa \in \{1,\ldots,k-1\}$, one label is dominated and can be deleted immediately.

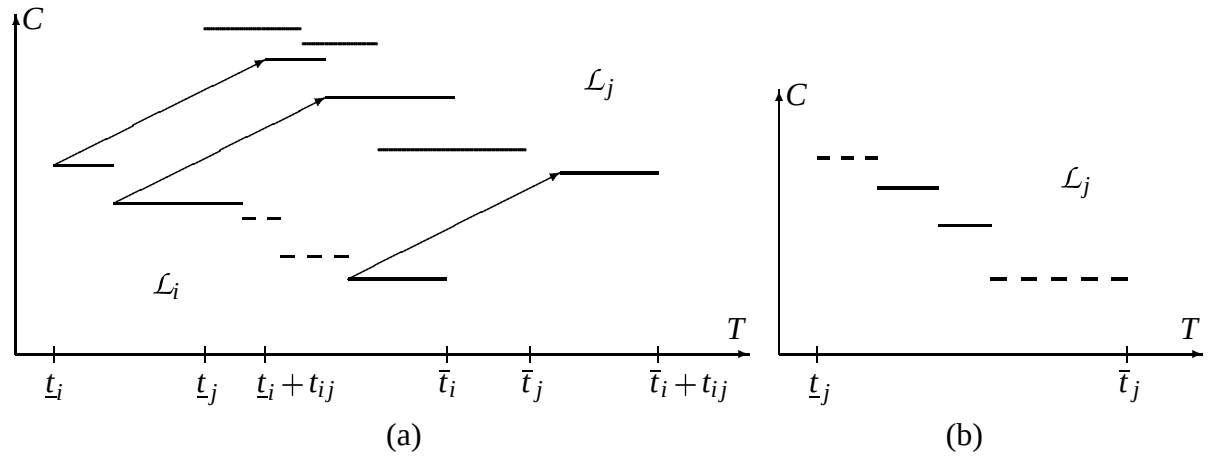

Figure 4.5: (a) Efficient treatment of labels at node i, with recent labels $\mathcal{L}_i'$ plotted as solid lines, whereas older labels $\mathcal{L}_i \setminus \mathcal{L}_i'$ are drawn as dashed lines. The translation $T_i + t_{ij} = T_j$ for each recent label is symbolized by an arrow. Part (b) shows the resulting $\mathcal{L}_j$ with dominated labels eliminated, recent labels $\mathcal{L}_j'$ again plotted solidly.

ferent nodes. There is a tendency that more often small amounts of labels are produced at each node, resulting in a frequent update of the respective lists of labels. An alternative is the *pulling* of labels (*see* IBARAKI (1987), and the references therein). Per iteration, new labels are generated at one node only, extending paths so as to terminate in that node. DESROCHERS & SOUMIS (1988*b*) first make use of this treatment in the context of the SPPTW.

DESROCHERS & SOUMIS (1988*a*) propose a different way of label treatment, *viz.* maintaining *all* labels in increasing order of time in *one* list only. Once, a label is treated, it is ensured by positive arc durations that it is never modified again, hence called *permanent*. The resulting label setting algorithm is a generalization of DIJKSTRA's. Using the concept of *buckets* to access the labels, their implementation is very efficient. In fact, these authors obtain the best possible pseudo-polynomial running time of $O(T^2)$, where $T = \sum_{i \in N} (\bar{t}_i - \underline{t}_i + 1)$.

Remark. Multiple visits to nodes are due to negative cost cycles, which are likely to be present in constrained shortest path problems arising as pricing problems. When positive arc durations are assumed, time windows hinder infinite traversals of negative cycles. Still, in practical implementations this problem is a severe performance issue, when *exact* solutions are sought, and time windows are wide compared to travel times.

Including Capacity, Pairing, and Precedence Constraints

In contrast to time window constraints, which are *local* at the nodes, pairing and precedence constraints require knowledge about *all* the nodes being visited on a path. PSARAFTIS (1980, 1983) develops an $O(|\mathcal{R}|^2 \cdot 3^{|\mathcal{R}|})$ dynamic programming algorithm for the single-vehicle dial-

a-ride problem with time windows (1-DARP), minimizing the total travel time. However, the proposed state space, and with it the algorithm's computational complexity, is independent of time window widths. We therefore consider here in more detail the extension of the above label correcting algorithm for SPPTW to the case with pickups and deliveries, basically worked out by three authors. While DESROSIERS, DUMAS & SOUMIS (1986) still solve the 1-DARP, where all nodes have to be visited, DESROSIERS & DUMAS (1988) carry the ideas to the 1-PDPTW. Their algorithm is already designed for solving the subproblem in a column generation context, which is finally suggested (DUMAS, DESROSIERS & SOUMIS 1991).

As before, we will first concentrate on the node disjoint version of the 1-PDPTW. We assume that not all nodes have to be visited. The objective is to minimize the total travel distance. Algorithm 4.9 then only needs some adjustments to the more general situation. Most notably, three-dimensional labels (R_i^k, T_i^k, C_i^k) represent a k^{th} feasible path which visits precisely the node set $R \subseteq N$, and ends in node $i \in R$. The other two entries of a label retain their former meaning. Note, that the vehicle load in node i can be calculated as $\sum_{r \in R_i^k} \ell_r$.

The consequence for the treatment of labels is immediate. Extensions of the k^{th} path by an additional arc $(i, j) \in \delta(i)$ are admissible if and only if capacity and precedence constraints are respected, and $j \notin R_i^k$. Paths cannot visit the (possibly virtual) destination node of a path unless also the pairing constraints are satisfied. The initial labels have $R_i^1 = \{i\}$ for all $i \in N$. From a complexity result by DROR (1994) is follows that the requirement for node disjoint paths renders the 1-PDPTW $\mathcal{NP}$-complete in the strong sense.

Relaxing this (and only this) requirement cannot simply be done by dropping again the third dimension of labels, which was introduced for the purpose of controlling the pickup and delivery constraints. Alternatively, in addition to storing the set $R \subseteq N$ of all visited nodes, we memorize for each path the set of visited *pickup nodes* $R^+ \subseteq N \cap \bigcup_{r \in \mathcal{R}} \{r^+\}$, the corresponding delivery nodes of which are still unvisited. Then, in Algorithm 4.9 the treatment of node i is replaced by the following.

$\vdots$
for all $(R_i^{+k}, T_i^k, C_i^k) \in \mathcal{L}_i$ **do**
  for all $(i, j) \in \delta(i)$ with $j \notin R_i^{+k}$ **do**
    if $T_i^k + t_{ij} \leq \bar{t}_j$ **then**
      if $j = r^+$ for some $r \in \mathcal{R}$ **and** $\sum_{r \in R_i^{+k} \cup \{j\}} \ell_r \leq L$ **then** // L is the vehicle capacity
        $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{(R_i^{+k} \cup \{j\},\ \max\{\underline{t}_j,\ T_i^k + t_{ij}\},\ C_i^k + c_{ij})\}$
      else if $j = r^-$ for some $r \in \mathcal{R}$ **and** $r^+ \in S_i^{+k}$ **then**
        $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{(R_i^{+k} \setminus \{r^+\},\ \max\{\underline{t}_j,\ T_i^k + t_{ij}\},\ C_i^k + c_{ij})\}$
      end if
    end if
  end for
  $\vdots$
end for
$\vdots$

We remark that pairing constraints are satisfied if and only if $R^+ = \emptyset$, and that for each label the set R is kept only for the purpose of reconstructing the corresponding path. Alternatively, a path may be stored in a different data structure. In what regards elimination of dominated labels, we have the following.

Lemma 4.10 (DESROSIERS & DUMAS 1988, DUMAS, DESROSIERS & SOUMIS 1991**)** *Given two labels* (R_i^{+k}, T_i^k, C_i^k) *and* $(R_i^{+k'}, T_i^{k'}, C_i^{k'})$ *with* $R_i^{+k} = R_i^{+k'}$, $T_i^k \leq T_i^{k'}$, *and* $C_i^k \leq C_i^{k'}$, *then label* $(R_i^{+k'}, T_i^{k'}, C_i^{k'})$ *can be eliminated.*

In order to check the dominance condition efficiently Lemma 4.10 suggests a two-level organization of the label data. At each node i, labels $(\cdot, T_i, C_i)$ with common first component R_i^+ are grouped in *states* (R_i^+, i). For each of these states, we can efficiently treat the corresponding (*de facto* two-dimensional) labels along the lines of Figure 4.5.

4.2.3 A Label Correcting Algorithm for ESPP

The ideas from the preceding subsection establish the algorithmic basis of our now to be developed shortest path algorithm which works directly on the pattern graph associated with an ESPP instance. The most fundamental distinction, suggested by the structure of $\mathcal{P}$-concatenations, is not to extend paths by nodes but by patterns taken from the family $\mathcal{P}^{1\cup 2}$.

When constrained shortest path problems are used to solve the column generator subproblem in the familiar set partitioning/covering type master program formulation as in our ESP context, the objective function value represents reduced costs, and arc weights are redefined as

$$\bar{c}_{ij} := \begin{cases} c_{ij} - u_r & \text{if } i = r^+ \\ c_{ij} & \text{otherwise,} \end{cases} \tag{4.17}$$

accounting for the dual variable value u_r corresponding to request $r \in \mathcal{R}$. Since we are to solve a separate pricing problem for each engine $e \in \mathcal{E}$, only those requests admissible on engine e are relevant, and will be denoted by $\mathcal{R}_e \subseteq \mathcal{R}$. The pattern graph $\mathcal{G}_\mathcal{P} = (\mathcal{P}^{1\cup 2}, \mathcal{A}_\mathcal{P})$ to be used in the algorithm is reduced accordingly.

Before proceeding we make some observations. To begin with, it follows from Lemma 1.3 on page 18 that, even though we have a pickup and delivery setting, pairing, precedence, and capacity constraints are always satisfied when we iteratively append patterns to (initially empty) $\mathcal{P}$-concatenations. Therefore, we only need to keep track about the remaining constraints, i.e., request disjointness and time windows. Secondly, the engine is stationed in a delivery location when decisions are due about further path extensions. This suggests grouping labels according to these nodes. Thirdly, the pattern graph may contain cycles C with $t_{ij} = 0$ on every arc $(i, j) \in C$, e.g., involving two local operations at the same track, *see* the examples on page 9. However, when this occurs, we always have strictly positive service times in at least one location. Consequently, again, paths are finite in presence of negative cost cycles, even if we relax the request disjointness requirement. Finally, Lemma 4.10 applies to our situation since we trivially maintain $R^+ = \emptyset$.

From these considerations we immediately derive our basic strategy, summarized in Algorithm 4.11 on the next page. We are already familiar with the principal procedure. With each delivery node i we associate states (R,i) and labels (T_i^k, C_i^k). These specify the k^{th} ESPP-concatenation which visits exactly requests $R \subseteq \mathcal{R}_e$, and lastly ends in node i. The start-of-service time there is T_i^k at a total reduced cost of C_i^k. A design alternative is based on states (R,P) with $P \in \mathcal{P}^{1\cup 2}$, which, in a sense, unduly decentralizes the information to be exploited in the application of Lemma 4.10, *see* also Figure 4.6 on page 107.

We keep labels in ascending chronological order. When time window infeasibility is detected while trying to expand a label, no further labels at the current state need to be treated—none will be feasible either. States are treated in chronological order as well. The pattern graph is not stored as defined, but as lists $\{(r^-, o(Q)) \mid (P,Q) \in \mathcal{A}_{\mathcal{P}},\ d(P) = r^-\}$, one for each delivery node r^-, $r \in \mathcal{R}_e$, reflecting the adjacency between delivery nodes and patterns which are admissible to be appended. These lists are sorted according to non-increasing dual variable values associated with the requests involved in the respective Q. Then, the algorithm tends to produce ESPP-concatenations with small reduced cost earlier, which is useful when the output of *any* negative reduced cost column suffices, *see* Subsection 4.2.6. Note, that label $(T_{e^+}^1, C_{e^+}^1)$ is initialized with $C_{e^+}^1 = -v_e$, the dual variable value associated with engine e. Alternatively, if desired, a fixed cost incurred by using this engine may be added.

We recall $d(\{e^+\}) = e^+$ and $s_{e^+} = 0$. All label lists $\mathcal{L}_{(R,i)}$ appearing in the course of the algorithm for the first time are initialized as empty sets. Request disjointness of paths is guaranteed by the condition $\text{requests}(Q) := \{r \in \mathcal{R}_e \mid r^+ \in Q\} \cap R = \emptyset$, checked when a label is treated. Note, that the way we stated Algorithm 4.11 allows to make use of pattern families different from $\mathcal{P}^{1\cup 2}$. Upon termination, the output is the column incidence vector

$$\boldsymbol{\delta}_{R^\star} \quad \text{at cost} \quad C_i^{k^\star} + \sum_{r\in R^\star} u_r + v_e, \quad \text{where } (R^\star, k^\star) = \operatorname*{arg\,min}_{(R,k)} \{C_i^k \text{ at state } (R,i)\} \tag{4.18}$$

if $C_i^{k^\star} < 0$, and otherwise is the information that no more columns with negative reduced cost exist. In the event that the pricing problem is required to return multiple columns, c.f. Section 5.2, we may evaluate (4.18) for each delivery location i. Heuristically, this leads to a certain *diversity* of the returned solutions.

Let us briefly point out an important distinction of our algorithm as compared to the one we discussed earlier for the general 1-PDPTW. Every generated label represents a feasible concatenation; we have a rolling planning horizon and there is no depot where the engines must return. That is, we *always* have a feasible solution at hand. This state of affairs will turn out helpful in Subsection 4.2.5 in reducing the number of states and labels.

This leads to an immediate algorithmic simplification when we relax the request disjointness requirement. Since no additional precautions have to be taken to satisfy the typical pickup and delivery constraints, we need not consider states at all. Instead, one reasonably stores the labels (T,C) directly associated to the nodes of the pattern graph, i.e., to the patterns themselves. This reduces the upper bound on the number of labels to only $|\mathcal{P}| \cdot \sum_{i^-\in\mathcal{N}} (\overline{t}_{i^-} - \underline{t}_{i^-} + 1)$. In the sequel, nonetheless, we exclusively consider the node disjoint ESPP.

Algorithm 4.11 Label Correcting Algorithm for ESPP

// Initialization
$\mathcal{L}_{(\emptyset,e^+)} \leftarrow \emptyset$
$\mathcal{L}'_{(\emptyset,e^+)} \leftarrow \{(T^1_{e^+} := \underline{t}_{e^+},\, C^1_{e^+} := -v_e)\}$
for all $r^-, r \in \mathcal{R}_e$ **do**
  $\mathcal{L}'_{(\emptyset,r^-)} \leftarrow \emptyset$
  $\mathcal{L}_{(\emptyset,r^-)} \leftarrow \{(T^1_{r^-} := \underline{t}_{r^-},\, C^1_{r^-} := \infty)\}$
end for
$\mathcal{T} \leftarrow \{(\emptyset, e^+)\}$
// Preprocessing of $\mathcal{R}_e$ and $\mathcal{A}_\mathcal{P}$, *see* Subsections 4.2.4, 4.2.5, and 4.2.6
// Treatment of states (R,i) in chronological order
while $\mathcal{T} \neq \emptyset$ **do**
  Choose state $(R,i) \in \mathcal{T}$ with $\underline{t}_i$ minimal
  // Possibly skip state (R,i) if $|R|$ larger than a threshold, *see* Subsection 4.2.6
  for all $(P,Q) \in \mathcal{A}_\mathcal{P}$ with $d(P) = i$ **and** requests$(Q) := \{r \in \mathcal{R}_e \mid r^+ \in Q\} \cap R = \emptyset$ **do**
    for all $(T^k_i, C^k_i) \in \mathcal{L}'_{(R,i)}$ **do**
      if new state $(R \cup \text{requests}(Q), d(Q))$ is promising **then** // *See* Subsection 4.2.5
        $j_1 \leftarrow i$
        $\tilde{T} \leftarrow T^k_i$
        $\tilde{C} \leftarrow C^k_i$
        for $j_2 = o(Q), \ldots, d(Q)$ **do** // consider all nodes in Q in order
          if $\tilde{T} + s_{j_1} + t_{j_1 j_2} \leq \overline{t}_{j_2}$ **then**
            $\tilde{T} \leftarrow \max\{\underline{t}_{j_2},\, \tilde{T} + s_{j_1} + t_{j_1 j_2}\}$
            $\tilde{C} \leftarrow \tilde{C} + \overline{c}_{j_1 j_2}$
          else
            Expansion is time infeasible; proceed to label elimination
          end if
          $j_1 \leftarrow j_2$
        end for
        if new label is not heuristically eliminated **then** // *See* Subsection 4.2.6
          $\mathcal{L}'_{(R\cup\text{requests}(Q),d(Q))} \leftarrow \mathcal{L}'_{(R\cup\text{requests}(Q),d(Q))} \cup \{(\tilde{T},\tilde{C})\}$
          $\mathcal{T} \leftarrow \mathcal{T} \cup \{(R \cup \text{requests}(Q), d(Q))\}$
        end if
      end if
    end for
    Eliminate dominated labels from $\mathcal{L}_{(R\cup\text{requests}(Q),d(Q))} \cup \mathcal{L}'_{(R\cup\text{requests}(Q),d(Q))}$
  end for
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(R,i)\}$
  $\mathcal{L}_{(R,i)} \leftarrow \mathcal{L}_{(R,i)} \cup \mathcal{L}'_{(R,i)}$
  $\mathcal{L}'_{(R,i)} \leftarrow \emptyset$
end while

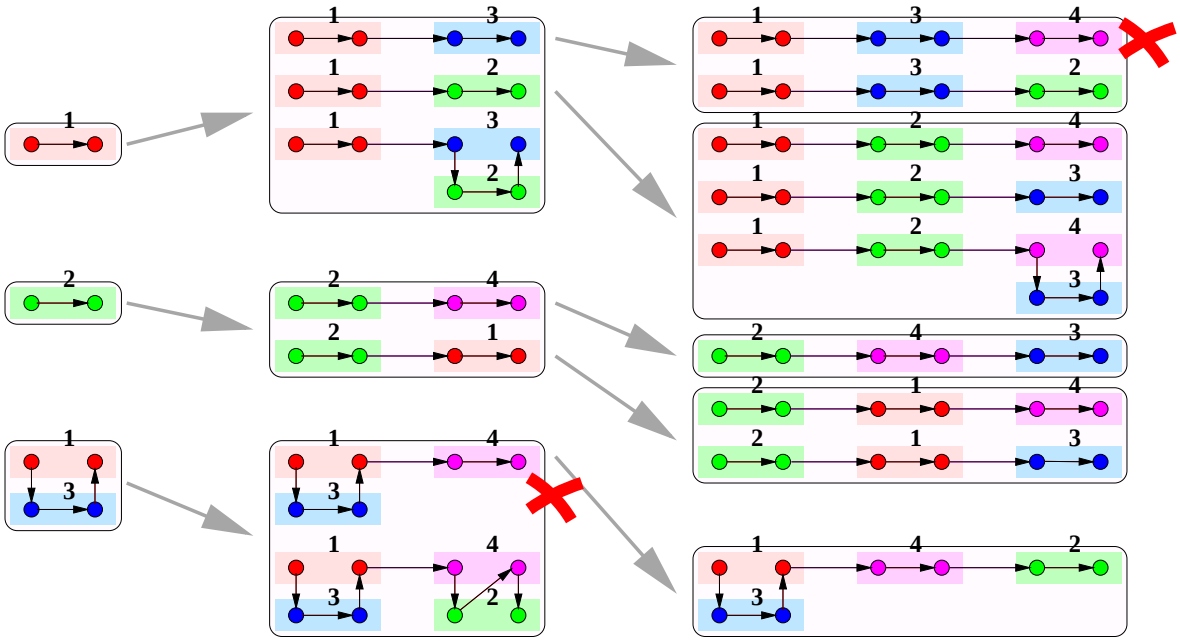

Figure 4.6: Extending concatenations in Algorithm 4.11. The marked concatenations are stored in the same state $(\{1,3,4\},4^-)$, thus allowing for an efficient elimination of a possibly dominated label.

4.2.4 Standard Refinements

The actual running time of all presented labeling algorithms obviously depends on the number of labels generated. In fact, the tree-like development of label expansions as suggested by Figure 4.6 gives an impression of the number of feasible concatenations constructed in Algorithm 4.11, provided no label elimination took place. At the k^{th} *stage*, we would have an additional number of at most $|\mathcal{P}| \cdot (|\mathcal{P}|-1) \cdot \ldots \cdot (|\mathcal{P}|-k+1)$ labels in the worst case. A straight forward upper bound on the total number of labels is $|2^{\mathcal{R}_e}| \cdot |\mathcal{R}_e| \cdot \max_{i^- \in \mathcal{N}}(\bar{t}_{i^-} - \underline{t}_{i^-} + 1)$.[4] Facing this tremendous exponential growth, one strives to reduce the number of labels by all (reasonably efficient) means. In this subsection we are concerned with exact methods only, whereas some heuristics are discussed in Subsection 4.2.6.

Constraint Based Preprocessing

It is intuitive reasoning that limiting the possibilities of feasible expansions of a label limits the growth of the number of labels. In essence, this boils down to reducing the pattern graph as much as possible by exploiting the problem constraints. *Preprocessing* deals with this topic *before* the algorithm is executed.

[4] More accurate numbers for a special case will be provided in Chapter 6.

Time window reductions, which are worthwhile by Definition 4.4 and reported effective in the literature, were already considered in Section 3.3. Here, we follow the lines of DESROCHERS, DESROSIERS & SOLOMON (1992) for a more extensive procedure. The following reductions are sequentially applied at each location, i.e., at each node $k \in \mathcal{N}$ of the request graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$:

1. $\underline{t}_k \leftarrow \max\{\underline{t}_k,\ \min_{(i,k)\in\mathcal{A}}\{\underline{t}_i + s_i + t_{ik}\}\}$
2. $\underline{t}_k \leftarrow \max\{\underline{t}_k,\ \min_{(k,j)\in\mathcal{A}}\{\underline{t}_j - s_k - t_{kj}\}\}$
3. $\overline{t}_k \leftarrow \min\{\overline{t}_k,\ \max_{(i,k)\in\mathcal{A}}\{\overline{t}_i + s_i + t_{ik}\}\}$
4. $\overline{t}_k \leftarrow \min\{\overline{t}_k,\ \max_{(k,j)\in\mathcal{A}}\{\overline{t}_j - s_k - t_{kj}\}\}$

Going over the locations, the criteria are cyclically applied until no more reductions take place. We remarked earlier that engines may differ in speed, from where different time window reductions arise for different engines. It can happen that $\underline{t}_i > \overline{t}_i$ for some $i \in \mathcal{N}$ after this procedure. Such nodes are eliminated. The consequence for the pattern graph is that nodes and arcs are discarded which cannot be part of any feasible solution.

Elimination Criteria at Runtime

Checking violated constraints is much more effective in the course of the algorithm because e.g., the actual arrival times at the nodes are known. Besides violated time windows, for the 1-PDPTW a multitude of elimination criteria was proposed by DESROSIERS, DUMAS & SOUMIS (1986). Not only the standard is probed, like pairing, precedence, and capacity constraints. The authors check whether all unvisited delivery locations corresponding to already visited pickup locations can be appended to a path in time or not. Since this requires the solution of a TSPTW the search is restricted to at most two unvisited delivery nodes.

Interestingly enough, nearly all of these conditions are useless for us, since by construction, labels bearing (actual or potential) infeasibilities are not constructed in the first place. Therefore, only two of their criteria remain which are independent of constraint violations.

Let two requests $r_1 \neq r_2 \in \mathcal{R}_e$ correspond to local operations at the same physical location, i.e., $c_{r_1^+ r_1^-} = c_{r_2^+ r_2^-} = c_{r_1^+ r_2^+} = c_{r_1^- r_2^-} = 0$. States $(R \setminus \{r_1\}, r_2^-)$ and $(R \setminus \{r_2\}, r_1^-)$ may be present simultaneously. For the same start-of-service time, identical labels with respect to cost are produced. Thus, we fix a precedence order of service on r_1 and r_2, according to increasing total service time of the respective request. *See* DESROSIERS, DUMAS & SOUMIS (1986) for a proof.

As indicated in Figure 4.6, the treatment of states need not be chronologically, but may as well be implemented in increasing order of the cardinality of R. Then, at stage s those concatenations are generated which visit precisely s requests, and labels from earlier stages may be eliminated when they are too expensive. However, this criterion should be considered only if memory usage is an issue.

Upper Bounds on Cost Coefficients

Another simple rule is applicable when upper bounds can be imposed on the (not reduced) cost of a concatenation. When, for example, many requests are geographically close, and the objective is to minimize dead heading, i.e., the connection distances between patterns, small cost coefficients are to be expected. Too expensive labels are then discarded.

4.2.5 Dual Variable Based Label Elimination

Feasibility based label elimination criteria proposed in the literature, except for time infeasibilities, are already inherently observed by our definition of $\mathcal{P}$-concatenations. We have also seen that efficiently checking the dominance relation depends on the implementation, and requires a good organization of the label space. Moreover, we can only compare labels *already constructed*. Rather, it would be desirable to have an *anticipatory prevention* of unpromising labels. We will now provide one such criterion.

Algorithm 4.11 and Label Elimination by a Lower Bound

At first, we recall to mind that the notion of *states* in the label correcting algorithms was not chosen accidentally, but to remind of their close relationship to the recurrence formula (4.14). In fact, the interpretation as dynamic programming algorithms gives rise to the results of this subsection.

Dominance among states is the classical means to reduce the solution space of a dynamic program. On the other hand, in branch-and-bound methods, a good performance essentially hinges on good bounds. The frameworks of these two well-known implicit enumeration strategies exhibit large similarities, and again, we refer to e.g., IBARAKI (1987) for supplementary reading. The common framework motivates us to investigate lower bounds in the context of reducing the number of labels in our dynamic program, *viz.* Algorithm 4.11. The generic idea is originally due to MORIN & MARSTEN (1976, 1978), and ALEKSEEV & VOLODOS' (1976), independently.

When solving ESPP, we are given a fixed engine $e \in \mathcal{E}$. With respect to (4.3), Algorithm 4.11 implicitly explores Ω_e, the set of admissible request sets for engine e. During the search, denote by $\overline{\Omega}_e \subseteq \Omega_e$ the subset so far considered, i.e., $R \in \overline{\Omega}_e$ if and only if we have already generated labels with finite cost for some state $(R, \cdot)$. Furthermore, we denote by C^{inc} the cost of a currently cheapest label, referred to as the *incumbent (value)*. Initially, $C^{inc} = \infty$. Let R^{inc} denote the associated admissible request set, i.e., for some $i \in R^{inc}$

$$C^{inc} \quad := \quad C^{i}_{R^{inc}} \quad = \quad \min \left\{ c^{e}_{R} - \sum_{r \in R} u_r - v_e \mid R \in \overline{\Omega}_e \right\} \quad .$$

Note, that C^{inc} is the optimal value sought by the procedure, when $\overline{\Omega}_e = \Omega_e$. Efforts to eliminate labels aim at precluding as large a subset of Ω_e as possible from being searched, while

guaranteeing that an optimal concatenation *will* be identified. Since ESPP is a pricing problem we are interested in *negative* values of C^{inc} only, and it would be helpful to know if the treatment of a given state (R,i) *can* eventually lead to a state with negative reduced cost labels at all. Even stronger is the question whether *any* future treatment of (successor states of) (R,i) exists that yields a better incumbent. A negative answer immediately enabled us to prune the search, excluding (R,i) from further treatment, and therefore reducing the searched state and label space. Originating from the branch-and-bound world, this procedure is sometimes referred to as *fathoming*. Clearly, as soon as

$$LB(R) \geq \min\{0, C^{inc}\} \tag{4.19}$$

holds for a lower bound $LB(R)$ on the best possible reduced cost coefficient obtainable by treatment of any label associated with states $(R,\cdot)$, we have such a negative answer. Labels associated with states *not* fulfilling (4.19) are called *promising*. All other labels are *unpromising* and will not be treated. Some additional notation will be useful throughout our discussion of (4.19). We denote by $\mathcal{R}_e^+$ the subset of requests admissible on engine e with associated positive dual variable value, i.e. $\mathcal{R}_e^+ := \{r \in \mathcal{R}_e \mid u_r > 0\}$.

Let $R \subseteq \mathcal{R}_e$ be the admissible request set corresponding to a particular state considered during the execution of Algorithm 4.11. The only way to lessen cost of the labels associated with this state is to expand the corresponding concatenations by requests in $\mathcal{R}_e^+$. Hence, a self-evident way of providing a lower bound to be used in (4.19) is

$$LB(R) \quad := \quad \min_{i \in R} C_i^{\hat{k}} - \sum_{r \in \mathcal{R}_e^+ \setminus R} u_r \ , \tag{4.20}$$

where the $\hat{k}^{\text{th}}$ label is last in the list of labels at state (R,i), which by (4.16) corresponds to the least cost label at state (R,i). The minimum can be easily updated and stored separately. Having to compute (4.20) for only one label among all labels associated with all states $(R,\cdot)$ facilitates the implementation of the proposed test, and renders it efficiently executable.

As a refinement, $C_i^k - \sum_{r \in \mathcal{R}_e^+ \setminus R} u_r$ could be kept for *each* label (T_i^k, C_i^k), thus possibly not disregarding entire states, but only shortening the respective lists $\mathcal{L}_{(R,i)}$ of labels. This individual bound

$$LB(R, C_i^k) \geq \min\{0, C^{inc}\} \tag{4.21}$$

for each label has the potential to eliminate more labels in the end.

We would like to remark, that independently of our work a very similar dual variable based lower bounding technique was developed by MEHROTRA & TRICK (1998). The authors propose a set partitioning based column generation algorithm for (a capacitated version of) the clique partitioning problem for graphs. However, it must be pointed out that their pricing algorithm actually is of branch-and-bound style, where the use of bounds is essential to the method. Most recently, MEHLHORN & ZIEGELMANN (2000) suggested a technique closely related to ours. As part of a solution procedure for the resource constrained shortest path problem such paths are constructed via dynamic programming. The authors eliminate unpromising paths, using a

reduced cost based lower bound. The dual variables are furnished by the dual optimal solution to the linear programming relaxation of a set partitioning formulation of the original problem.

Obtaining a lower bound from dual variables and exploiting it in the context of dynamic programming used for a pricing problem is a new development, and to the best of our knowledge, is proposed here for the first time.

Also related is a paper by CRAINIC & ROUSSEAU (1987). In a set partitioning formulation of the airline crew scheduling problem they find good pairings, i.e., columns, by judging column quality by the dual variable values of the components the columns are made of. By exploiting the problem structure, the authors manage to reduce the intervals within which the dual variable values are allowed to range. MINGOZZI, BIANCO & RICCIARDELLI (1997) use sophisticated lower bounds to reduce the state space of a dynamic program to solve the TSP with time windows and precedence constraints. Their results, however, are not in the column generation context, but if they were, could be combined with ours, possibly leading to a performance improvement.

Improving the Lower Bound

The choice of $\mathcal{R}_e^+$ in the derivation of $LB(R,C_i^k)$ involved the relaxation of time window constraints, and ignored patterns. In the sequel we explain some possibilities to strengthen the lower bound by choosing more *meaningful* request sets $\mathcal{R}_e^\oplus \subseteq \mathcal{R}_e^+$, which substitute for $\mathcal{R}_e^+$. We find such $\mathcal{R}_e^\oplus$ by respecting at least a subset of the previously relaxed time window constraints.

One obvious drawback of the above lower bounds is their pessimism, since hardly *all* unvisited requests in $\mathcal{R}_e^+$ will be appended to a given concatenation. Possibly known upper bounds on the maximal cardinality of an admissible request set may be exploited here. Another technique to reduce $\mathcal{R}_e^+$ is the following. We call two requests $r_1 \neq r_2 \in \mathcal{R}_e^+$ *incompatible* on $e \in \mathcal{E}$ with respect to a given label (T_i^k, C_i^k) at state (R,i) if and only if for e no admissible superset of R can contain both, r_1 and r_2, simultaneously. That is, $R \cup \{r_1,r_2\} \not\subseteq R' \in \Omega_e$ under the premise that the concatenation corresponding to R conforms to (T_i^k, C_i^k). Otherwise r_1, r_2 are called *compatible*. Conflicting time windows may be a reason for incompatibility. Then,

$$\mathcal{R}_e^\oplus = \mathcal{R}_e^+ \setminus \{\arg\min\{u_{r_1}, u_{r_2}\}\} \tag{4.22}$$

strictly improves $LB(R,C_i^k)$: Even if serving $\arg\min_{r\in\{r_1,r_2\}} u_r$ incurs zero cost, the corresponding dual variable has strictly positive value. The property of incompatibility does not depend on request sets alone, because it is not invariant under taking supersets. Nevertheless, incompatible requests for a given label retain this property for all labels produced by treatment of this label, when incompatibility is caused e.g., by violated time windows.

More generally, we define an incompatible request set $I \subseteq \mathcal{R}_e^+$ with $|I| \geq 2$ in the canonical way, i.e., any two requests $r \neq s \in I$ are incompatible. Analogously to (4.22), we reduce

$$\mathcal{R}_e^\oplus \leftarrow (\mathcal{R}_e^\oplus \setminus I) \cup \{\arg\max_{r\in I} u_r\} \tag{4.23}$$

for *every* incompatible request set I. Note, that $|R \cap I| \leq 1$ since R corresponds to at least one

feasible concatenation. Moreover, $|\mathcal{R}_e^{\oplus} \cap I| \leq 1$, and $|I_1 \cap I_2| = 0$ for any two incompatible request sets $I_1 \neq I_2$, c.f. Figure 4.7. Therefore, each application of (4.23) strengthens $LB(R, C_i^k)$.

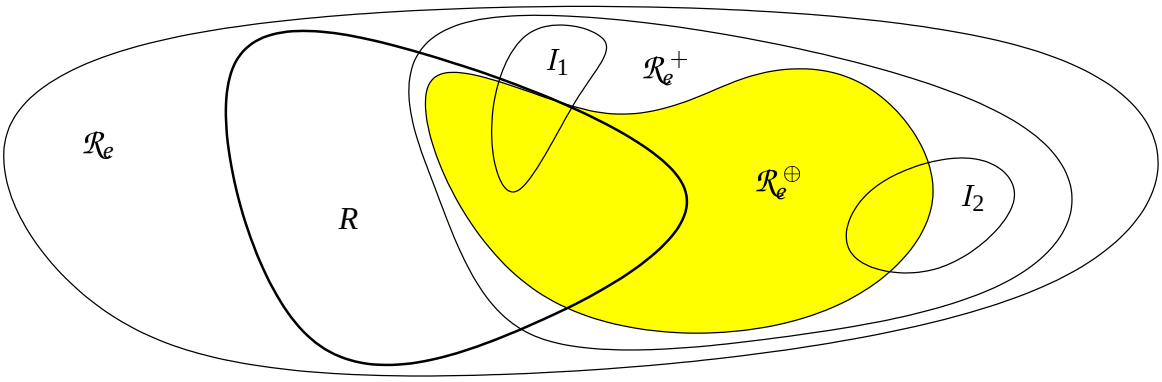

Figure 4.7: Possible inclusions of request sets considered for the lower bound $LB(R, C_i^k)$

Intuitively, (4.23) is most effective for a *maximally incompatible* request set I^{max}. That is, each $r \in (R \cup \mathcal{R}_e^+) \setminus I^{max}$ is compatible with at least one $i \in I^{max}$. However, already the problem of determining the cardinality of I^{max} is prohibitive from a computational standpoint. The answer $|I^{max}| = 1$ is equivalent to the existence of a feasible $\mathcal{P}^1$-concatenation on the unvisited requests in $\mathcal{R}_e^+$. Deciding this latter question is $\mathcal{NP}$-complete in the strong sense by Lemma 1.8.

Pattern Exclusion and Preprocessing

Finally, for a particular label (T_i^k, C_i^k), adjoining single patterns $P \in \mathcal{P}$ may be unprofitable, when the requests $r_1, \ldots, r_k$ visited by P belong to $\mathcal{R}_e^- := \{r \in \mathcal{R}_e \mid u_r < 0\}$. Pattern P cannot belong to any ESPP-concatenation constructed from (T_i^k, C_i^k) if

$$C_i^k - \sum_{r \in \mathcal{R}_e^{\oplus} \setminus R} u_r - \sum_{r \in \{r_1, \ldots, r_k\}} u_r \quad \geq \quad \min\{0, C^{inc}\} \quad . \tag{4.24}$$

The special case $R = \emptyset$ can be applied for a preprocessing in Algorithm 4.11. If (4.24) holds for the initial label $(T_{e^+}^1, C_{e^+}^1)$ at state $(\emptyset, \{e^+\})$, no pattern that visits $r_1, \ldots, r_k \in \mathcal{R}_e^-$ is profitably contained in any ESPP-concatenation and can be discarded throughout the procedure. The same test is applicable to patterns visiting r_1, r_2 with $u_{r_1} > 0$ and $u_{r_2} < 0$.

Remark. EASTON (1990) points out that a poor incumbent C^{inc} may result in a considerable computational burden, possibly larger than not using a fathoming criterion at all. Even though we also check our bounds against zero, these concerns are valid. EASTON proposes to use the relative success in fathoming labels as a measure for incumbent quality. That is, when relatively few labels are fathomed, e.g., a heuristic may be called to furnish a better incumbent. Precautionary, we try to provide a good initial incumbent by means of heuristics, c.f. Subsection 4.2.6.

Remark. We revisit the two phase method of finding initially admissible request sets presented in the previous section on page 88. Not exclusively, but also in the context of our lower bounds, choosing the penalty cost M as small as possible is important: As long as artificial variables are basic a large M contributes to large dual variable values, possibly weakening the lower bound.

Further Improvements

In constructing $LB(R,C_i^k)$ we have been considering dual variable values only and ignored the original objective function completely. Various extensions are conceivable, one of which is augmenting $LB(R,C_i^k)$ by (a lower bound on) the minimal cost incurred by adding unvisited requests $r \in \mathcal{R}_e^{\oplus} \setminus R$ to the current concatenation. That is,

$$LB(R,C_i^k) + \min_{r \in \mathcal{R}_e^{\oplus} \setminus R} \left\{ c_{ir^+} + \min_{P \in \mathcal{P}} \left\{ \sum_{(l,j) \in \mathcal{A}|_P} c_{lj} \mid o(P) = r^+,\ \text{requests}(P) \subseteq \mathcal{R}_e^{\oplus} \setminus R \right\} \right\}$$

improves upon $LB(R,C_i^k)$ by the minimum cost to reach and traverse the next pattern. Unlike other resource constrained shortest path problems, the shortest distance from any delivery node i to the (virtual) terminal node e^- is always zero, hence cannot be used for further enhancement.

The question for an optimal lower bound at each label (T_i^k, C_i^k) naturally emerges from a theoretical viewpoint. LAGRANGIAN relaxation plus a subgradient method is an option here, and was suggested for the calculation of bounds in dynamic programming algorithms e.g., by BEASLEY & CHRISTOFIDES (1989), and DYER, RIHA & WALKER (1995). However, in what regards the goal of this chapter, *viz.* actually *solving* the ESPP on a computer, this question leads us astray. Having to treat labels several thousand times in one execution of Algorithm 4.11, benefits gained from excellent but computationally costly label elimination criteria may be outweighed by inadmissible running times. This tradeoff is classically known from the literature, and brought us to not further follow this thread.

Remark. It is important to see that without modification our lower bound is suited for request disjoint concatenations only, since multiple visits to the same request multiply incur the corresponding dual cost. I would like to thank BRIAN KALLEHAUGE who pointed this out to me.

4.2.6 Heuristics

An exact algorithm for ESPP is indispensable when a certificate of optimality is needed upon completion of the column generation algorithm. However, especially in the beginning, a pricing algorithm may return *any* column with negative reduced cost coefficient. Heuristics are a way to do this fast. The greedy and arc exchange heuristics discussed in Subsection 4.1.1 for the construction of feasible $\mathcal{P}^{1\cup 2}$-concatenations can be plainly adapted by using the adequate reduced cost structure reflected by (4.17). They are not repeated here. Instead, we restrict our discussion to heuristic variants of the label correcting Algorithm 4.11. That is, we waive optimality by different restrictions on its solution space. These variants are separately presented in turn, but can be arbitrarily combined.

Premature Termination Perhaps the most immediate modification is to return the column corresponding to the first, or more generally, to the k^{th} negative reduced cost label constructed, with a given parameter $k \in \mathbb{N}$. This parameter controls the trade-off between solution quality and computation time. Beyond being very easy to implement, this idea is attractive for its implicitly *dynamic* behavior. Without further algorithmic efforts the method finally delivers an optimal solution as well.

Cost Coefficient Improvement For every heuristically generated column incidence vector $\boldsymbol{\delta}_R$, the respective cost coefficient c_R^e is not necessarily minimal. This implies, that the same column is potentially regenerated with smaller cost coefficient in a later column generation iteration. When $|R|$ is not too large, say $|R| \leq 4$, it is advantageous to calculate the best possible c_R^e by determining the optimal ESPP concatenation for the fixed request set R. The backtracking Algorithm 4.1 of Subsection 4.1.1 (with c_{ij} replaced by $\bar{c}_{ij}$ for all $(i,j) \in \mathcal{A}$) is suited to achieve this.

Pattern Graph Reduction We have observed earlier that the size of the pattern graph directly influences the number of generated labels. DESROSIERS, DUMAS & SOUMIS (1986) propose to heuristically reduce the number of nodes and arcs. Expensive requests $r \in \mathcal{R}_e$ with $u_r < \alpha \in \mathbb{R}$ and expensive connections $(i,j) \in \mathcal{A}$ between two locations with $c_{ij} > \beta \in \mathbb{R}_+$ are deleted from the request graph. Thus, patterns and the incident arcs are eliminated from the pattern graph. This proceeding is similar to the one suggested by DUMAS, DESROSIERS & SOUMIS (1991). The authors successively use reduced graphs which contain the cheapest 30%, 50%, and finally 100% of the arcs. Hence, again, an optimum to the pricing problem is obtained in the end.

When only full truckload patterns, i.e., $P \in \mathcal{P}^1$ are considered, we reduce the number of nodes by an order of magnitude. Moreover, this heuristic usually produces feasible solutions because it reflects current manual planning. Similar to a proposal by CRAINIC & ROUSSEAU (1987) we may alternatively drop patterns $P \in \mathcal{P}^2$ involving $r_1, r_2 \in \mathcal{R}_e$ with $\sum_{(i,j)\in\mathcal{A}|_P} c_{ij} - u_{r_1} - u_{r_2} \geq 0$.

Request Aggregation and Decomposition Two further reduction techniques benefit from a certain knowledge about the proximity of two patterns, e.g., temporal or geographical. Request disjoint patterns which are close with respect to the used measure are grouped together. Cost and travel time are assigned to each group, e.g., the reduced cost incurred and the duration when traversing the contained patterns in a fixed, or even an optimal order. This grouping is generically known as aggregation. Only arcs incident to the groups' terminal nodes are kept, and the algorithm works on this smaller graph, considering groups as large patterns.

Another strategy is to decompose the entire set of requests into smaller, not necessarily disjoint subsets. The pricing problem is then solved on the pattern graph induced by one of these subsets only. The subsets may e.g., be considered cyclically in successive calls to the algorithm. Observe the resemblance to our decomposition of the planning horizon on page 92. JOHNSON, MEHROTRA & NEMHAUSER (1993) suggest the notion of *subset column generation* .

Excessive Fathoming Suppose a label (T_i^k, C_i^k) is fathomed when

$$\gamma \cdot LB(R, C_i^k) \geq \min\{0, C^{inc}\} \quad \text{with} \quad 0 < \gamma < 1. \tag{4.25}$$

The case $\gamma = 1$ is identical to our investigations in Subsection 4.2.5, and hence gives an *exact* algorithm. On the other hand, $\gamma = 0$ renders (4.25) redundant, leading to an elimination of all labels. Assume that $LB(R, C_i^k) \leq 0$, for otherwise the multiplication by γ would have no effect on the criterion. Then, for $0 < \gamma < 1$, we have $\gamma \cdot LB(R, C_i^k) > LB(R, C_i^k)$, and *more* labels are eliminated than before. In other words, an incumbent possibly with $C^{inc} > z_e^\star$ will be considered optimal.

Note, that $(z_e^\star - C^{inc})/z_e^\star < 1$ always holds for any incumbent with $0 > C^{inc} \geq z_e^\star$. That is, we have a trivial upper bound on the relative error incurred for *any* pricing heuristic. Most notably, the heuristic just described allows a better approximation guarantee.

Lemma 4.12 (Approximation Guarantee for Excessive Fathoming)
Let $z_e^\star < 0$. *When* (4.21) *is replaced by* (4.25) *with* $0 < \gamma \leq 1$, *then* $(z_e^\star - \min\{0, C^{inc}\})/z_e^\star \leq 1 - \gamma$.

Proof. The case $z_e^\star \geq 0$ is uninteresting since Algorithm 4.11 truly returns that no columns with negative reduced cost exist. Without loss of generality we restrict attention to $C^{inc} \leq 0$, since then $C^{inc} = \min\{0, C^{inc}\}$. Criterion (4.25) is more effective, i.e., eliminates more than (4.21) only if $LB(R, C_i^k) < C^{inc} \leq \gamma \cdot LB(R, C_i^k)$ for some label (T_i^k, C_i^k). When such an additional elimination takes place, we obtain for the incumbent

$$C^{inc} - z_e^\star \;\leq\; \gamma \cdot LB(R, C_i^k) - z_e^\star \quad \Longleftrightarrow \quad \frac{C^{inc} - z_e^\star}{z_e^\star} \;\geq\; \gamma \cdot \frac{LB(R, C_i^k)}{z_e^\star} - 1 \;\geq\; \gamma \cdot 1 - 1 \;\; .$$

The last inequality follows from $LB(R, C_i^k) \leq z_e^\star < 0$, which holds by definition of the lower bound. Constraining C^{inc} to be non-positive then immediately yields the claim. □

Alternative Treatment of States Finally, we present some heuristic design alternatives for the treatment of states in Algorithm 4.11. Instead of considering states (R, i) in chronological order, a possible treatment is in non-decreasing order of out-degree of node i in the pattern graph. More precisely, nodes are ordered according to non-decreasing $\sum_{P \in \mathcal{P}, d(P) = i} |\delta(P)|$. The rationale behind this modification is that label elimination criteria are most effective when applied early in the search, thus pruning large portions of the tree in Figure 4.6. This is the more likely the farther from the root the *fanout* appears (REINGOLD, NIEVERGELT & DEO 1977). Alternatively, we may order states (R, i) according to non-increasing value of the dual variable associated to the request with delivery location i. The motivation here is to construct good incumbents early.

A different strategy aims at the heuristic elimination of all states with $|R| \leq \rho$, where $\rho \in \mathbb{N}$ is a preset parameter. For instance, $\rho = \lceil |\mathcal{R}|/|\mathcal{E}| \rceil$ is a reasonable choice, when the request load on the engines is supposed to be almost balanced. This cardinality restriction in turn improves $LB(R, C_i^k)$, since only the $|R| - \rho$ most positive dual variables u_r, $r \in \mathcal{R}_e^\oplus \setminus R$ need to be considered for its calculation. The parameter ρ may be dynamically increased.

4.2.7 Re-optimization

Successive calls to the pricing algorithm for the same engine ask for solving similar problems, which differ only in their cost structure because of new dual variable values. This suggests the idea that building on solutions from prior calls may facilitate the problem. SPIRA & PAN (1975) showed that this re-optimization problem for the *un*constrained shortest path problem has the same computational complexity as solving the problem from scratch. Because of Algorithm 4.6 this result immediately extends to the ESPP. However, as the authors point out, these findings refer to the situation that only an optimal path is known to the re-optimization. Improvements are likely when intermediate information is kept on how this path was obtained.

In case of Algorithm 4.11 this construction information is present in form of states (R,i) and labels (T_i^k, C_i^k). Given the vectors $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$, and $\boldsymbol{u}, \boldsymbol{v}$ of current and former dual variables, respectively, this information is easily updated via $\bar{C}_i^k = C_i^k + \sum_{r\in R}(u_r - \bar{u}_r) + v_e - \bar{v}_e$ for each label. Subsequently, dominated labels are eliminated. There is no conclusion whether this proceeding results in less or even more generated labels for the re-optimization problem. After all, concatenations corresponding to negative $\bar{C}_i^k$, if any, yield feasible heuristic solutions.

When two consecutive executions of Algorithm 4.11 are considered for *different* engines, the admissible current and former request sets $\mathcal{R}_{\bar{e}}$ and $\mathcal{R}_e$, respectively, may differ as well. Therefore, labels may correspond to infeasible concatenations, unless $\mathcal{R}_{\bar{e}} \supseteq \mathcal{R}_e$. In the event that $\mathcal{R}_{\bar{e}} \subset \mathcal{R}_e$, the cost of infeasible labels at least give raise to lower bounds on the cost of feasible concatenations associated to the respective state. DESROCHERS & SOUMIS (1988*b*) exploit this knowledge in a re-optimization algorithm for the SPPTW. We do not deepen these considerations in this thesis.

4.3 Price-and-Branch

The potential problems concerning the existence of an integral solution for the optimal restricted master program are already stated clearly in Section 2.6. Our computational experiments indicate, under conditions to be explained in the next chapter, that the restricted master program happen to be already integer feasible, and the solution quality is acceptable. The column generation process is not invoked at any node other than the root node of the branch-and-bound tree. This policy may be termed *price-and-branch* by analogy with the folklore notion *cut-and-branch* where valid inequalities are adjoined to the LP at the root node only.

Here again, we come across the phenomenon that reality is not as bad as suggested by computational complexity, or in other words, in practice the theoretically worst case rarely occurs. This was observed in many other practical settings as well where artificial data were compared to “real world” data. Although not being a mathematically satisfactory explanation this is an encouragement to attack practical problems even in the presence of negative complexity results. Of course, in a failsafe industrial implementation one should safeguard against infeasibilities by including a branch-and-price code as emergency reserve. Also, one could invoke this code by default if there is time left until the scheduling decision is due in order to improve on the solution found by the price-and-branch approach.

CHAPTER 5

Implementation Issues

If you know the optimum, it is much easier to find.
—JACQUES DESROSIERS (1999)

In this chapter we describe in detail the implementation of our price-and-branch approach to the engine scheduling problem, making a few general remarks on implementing column generation codes as we proceed, when appropriate. Efficiency is of prime importance when practical problem sizes are to be attacked. It is our experience, complying to other authors, that even our profound methodological presentation in Chapter 2 still offers a vast degree of freedom from a computational standpoint. Most recently, DESAULNIERS, DESROSIERS & SOLOMON (1999) presented a compendious collection of implementation and acceleration techniques. Although sketchy, we believe that presently, their review can hardly be outmatched in terms of experience and scope. It should be definitively considered when it comes to an implementation.

In order to allow for an easy automatic testing of our proposals, we parametrized our code. Table 5.1 on the following page overviews the commandline options available to influence the program behavior. Most options are introduced throughout the chapter, otherwise are self explanatory.

5.1 Restricted Master Program

We introduced in Chapter 4 the time window reduction technique to *preprocess* the input data at the outset of the algorithm. Although still useful, we typically experienced a reduction by a few

Option	Default	Comment
`-c`	`false`	use set covering instead of set partitioning formulation
`-d`	`false`	enable deepest-cut pricing
`-dc` *red*	∞	delete columns with reduced cost $\bar{c} >$ *red* from restricted master
`-D`	`false`	output extensive debug information
`-e` *exp*	∞	expand each label in Algorithm 4.11 at most *exp* times
`-f`	`false`	each request is forced to be served full truckload
`-g`	`false`	produce a `GAMS` input file with model (ESPMIP), and exit
`-h`	`false`	print a help message, and exit
`-Ha`	`false`	artificial initial basis only, no greedy initial heuristics
`-Hc`	`false`	optimize with `-c` enabled, then disable `-c`, and re-optimize
`-Hd` *its*	0	apply dual heuristic for *its* master iterations
`-He`	`false`	initial basis generated by greedy engines heuristic
`-Hf`	`false`	optimize with `-f` enabled, then disable `-f`, and re-optimize
`-Hi` *inc*	0.0	increase value given by `-Hp` by *inc* in each master iteration
`-Ho`	`false`	turn off greedy pricing heuristic
`-Hp` *per*	1.0	consider only "best" $100 \cdot$ *per* percent of requests in Algorithm 4.11
`-Hr`	`true`	initial basis generated by greedy patterns heuristic with $\mathcal{P} = \mathcal{P}^1$
`-I`	`false`	allow tiny primal infeasibilities in restricted master
`-l` *eng*		specify number of engines
`-L`	`false`	disable label elimination by bound (4.19) in Algorithm 4.11
`-m` *hor*	∞	truncate time windows at *hor*; maximal length of planning horizon
`-M` *M*	1000	set "big *M*" used to penalize artificial variables
`-pa` *add*	1.0	add all columns with $\bar{c} <$ *add* from pool to the restricted master
`-po`	`false`	disable usage of the column pool
`-pt` *thr*	10.0	keep columns in column pool with $\bar{c} <$ *thr*
`-P` *k*	100	Algorithm 4.11 prematurely terminates after construction of the k^{th} negative reduced cost column; $k = 0$ disables premature termination
`-r` *req*		specify number of requests
`-s`	`false`	enable steepest-edge pricing
`-S`	`false`	enable a stabilized column generation in spirit of Subsection 2.5.4
`-T` *ub*	∞	generated columns in Algorithm 4.11 must satisfy $c \leq$ *ub* (not $\bar{c}$!)
`-v`	`false`	verbose mode

Table 5.1: Overview of commandline options. We also state our defaults, although it should be noted that differing settings do yield significantly better results for single problem instances.

percent only. In this sense our practical data is rather difficult. The pattern graph typically still exposes a density of more than 40%.

The objective function we actually implemented in our column generation code is minimization of the total waiting and dead heading time of all engines, i.e., the total time the engines drive empty or wait until time windows open. For this particular objective, the cost coefficients range in small integers for some practical instances, because *highly productive* solutions are feasible. Then, not only the restricted master program is degenerate, but so is its dual. Note also, that the cost for a concatenation cannot simply be calculated by straight forward addition of arc weights. Algorithm 5.1 outlines the master iteration of our price-and-branch heuristic, which we present in more detail below.

Algorithm 5.1 Price-and-Branch Heuristic for ESP, Master Iteration	relevant options
Process options	`-D -g -h -v`
Read data	`-l -m -r`
Initialize data structures, solver, and memory management	
Initialize column pool	`-pa -po -pt`
Relaxation or restriction of problem	`-c -f`
Preprocessing, enable heuristics	`-Hc -Hf -Hp -Hi -P`
Always add artificial basis	`-M`
Enable stabilization	`-S`
Construct initial heuristic solution	`-Ha -He -Hr`
Apply dual heuristic	`-Hd`
repeat	
for each engine $e \in \mathcal{E}$ **do**	
Call Algorithm 5.2 for engine e	`-d -e -L -s -T`
Delete columns with "large" reduced cost from restricted master	`-dc`
Re-optimize restricted master program	`-I`
end for	
if no columns price out favorably **and** heuristics are still enabled **then**	
Disable all heuristics	
end if	
until LP optimality is proved **or** stop rule (e.g., maximal computation time) applies	
Attempt to integrally solve the restricted master by branch-and-bound	
Output human readable schedule	
Free memory	

We always initialize the restricted master program with an all artificial basis as stated in (4.2). This is the default initial solution when the `-Ha` option is given. The artificial variable penalty M can be modified via `-M`. In the early stages of the algorithm, this penalty cost strongly influences the dual variable values. Therefore, this value should be carefully chosen. In our experience, very large penalties amplify the aforementioned effect; too small penalties fail to penalize sufficiently. Alternatively, we may activate the greedy engines or greedy patterns heuristic, respectively, as

described on page 89 with options `-He` and `-Hr`. For both heuristics only full truckload patterns are greedily concatenated.

Some additional modifications are sampled. It is evaluated whether it pays to have an excellent dual solution at hand. With option `-Hd` a precomputed dual optimal solution is periodically passed to the pricing problem in lieu of the current dual solution. It appears that these pricing problems become considerably harder so solve. In our experiments the overall effect is mixed.

We concluded in Subsection 2.4.1 that a restriction of the dual variable space is advantageous from a theoretical point of view. For this reason, e.g., VANDERBECK (1994) opts for using a set covering formulation instead of a set partitioning formulation, whenever applicable. The former may be enabled with option `-c`. However, we have $R_1 \subset R_2 \subseteq \mathcal{R} \not\Rightarrow c_{R_1} < c_{R_2}$ when the above objective function is used. We therefore cannot conclude that an optimal solution to the relaxed (covering) formulation optimally solves the original problem as well. Actually, we obtain strictly smaller optima in our experiments. In contrast, with option `-f` a *restriction* of the original problem may be considered, *viz.* adding the constraint that each request must be served full truckload, i.e., only patterns in $\mathcal{P}^1$ are admissible. In addition, the respective modification may be enabled, the restricted master solved to optimality, the unmodified formulation restored, and finally optimally solved. Use options `-Hc` and `-Hf`, respectively.

In order to allow for slightly increased degree of freedom when solving the restricted master program, we experimented with primal infeasibility, i.e., $-5.0 \cdot 10^{-4} \leq \lambda_q \leq 1.0$, $q \in Q'$. At LP optimality, the lower variable bounds are reset to 0.0, and the dual simplex method is called, before column generation proceeds as usual. This feature is enabled with option `-I`.

Finally, in addition to column generation also *column elimination* has been proposed in the literature, *see* e.g., JAUMARD, MEYER & VOVOR (1999) for an elaborate description. In fact, the idea of removing non-basic variables from the restricted master dates back to the days when scarce core memory was an issue. With option `-dc` a threshold for the reduced cost of a column can be given, above which the corresponding variable is eliminated. This parameter must be carefully chosen since it may cause the simplex algorithm to cycle. LINDEROTH (1998) additionally proposes to accommodate all columns deleted from the restricted master in a *column cache*, for potential further use.

5.2 Subproblem Solution and Column Management

The pricing subproblem is the most frequently executed essential component of a column generation code. Each call should therefore be as effective as possible, or in other words, the invested computation time should pay off to the largest possible extent. Even though this statement seems obvious it deserves some recognition. This becomes clear when we recall that it is neither mandatory to add a most profitable column to the restricted master program nor are we restricted to add only one column at a time—while this is precisely what happens if the pricing problem is solved exactly, in alternation with the re-optimization of the restricted master.

Already in Section 4.2 we tailored various algorithmic ingredients to our particular subprob-

lem ESPP for the sake of efficiency. Here, we are concerned with *synthesizing* these components to an overall column generator procedure. This merger is not unique. Not only the heuristics we discussed but also our exact label correcting algorithm exhibit a large degree of freedom in their actual realization and combination. It is our experience that these latter aspects outweigh the still valid importance of an efficient implementation of the respective components.

The global guideline is to call our exact pricing Algorithm 4.11 as seldom as possible. Other authors share this point of view, but in our case this is all the more important inasmuch we have to solve a *node disjoint* constrained shortest path problem, and this requirement is often not present in similar situations. The following strategy is adopted. We enable the *premature termination* of Algorithm 4.11, i.e., we stop as soon as the k^{th} negative reduced cost column is found, where the parameter $k \in \mathbb{N}$ is controlled via commandline option `-P` as listed in Table 5.1.

Column Pool

Still, (not only) negative cost columns are produced which are *not* added to the restricted master program according to this policy. The efforts for generating them appear to be in vain. Instead, we would store all columns with reduced cost smaller than a threshold to a so called *column pool*. This concept is well known from branch-and-cut codes (*see* JÜNGER & THIENEL 1998), and has been successfully applied in column generation implementations as well, *see* e.g., SOL (1994). Before any other construction method is invoked we check whether the pool contains columns which price out negatively. It may also be helpful to add columns to the restricted master with a small positive reduced cost. These may become active in later iterations. However, such columns are only added when in the same iteration at least one negative reduced cost column is added from the column pool. Otherwise, cycling could occur.

The pool is linearly searched. We perform for each entry (a) an update of reduced cost according to the current dual variables, (b) the addition of the column to the restricted master (and deletion from the pool), if applicable, and (c) a deletion of the column from the pool if it is too expensive. All thresholds are parameter controlled, c.f. Table 5.1. We neither limit the pool capacity nor the number of columns to be added to the pool or to the restricted master per iteration. SAVELSBERGH & SOL (1998) propose the selection of partial solutions, i.e., a set of columns which cover each request at most once, and originate from different engines. In our experiments this appears not necessary in terms of integer feasibility. However, we try to keep a pool of high quality, i.e., when a column is pushed to the pool we check whether its reduced cost can be improved in reasonable time by backtracking, c.f. Algorithm 4.1.

Greedy Dual Variable/Minimal Time Window Slack Heuristic

When no columns are delivered from the pool we invoke a very simple heuristic which greedily assigns full truckload patterns to the engine in question as long as this is time feasible. Each decision should provide a good decrease of the tentative column's reduced cost coefficient but maintain time window feasibility as long as possible. That is, a compromise is sought between

a large positive dual variable value and the least detriment of the degree of freedom for the remaining decisions. We therefore choose the next request to be assigned as a maximizer of the following ranking. Divide the dual variable value associated to a request by the maximal number of minutes the engine could be delayed while still completing the request in time. In other words, a critically small time window slack amplifies the dual variable weight of a request. *See* Algorithm 5.2 for a more formal description. The parameter $0 < \varepsilon \ll 1$ is chosen in order to avoid a division by zero.

Premature Termination of the Exact Label Correcting Algorithm

When also the above greedy heuristic fails to furnish a negative reduced cost column we invoke the prematurely terminating label correcting Algorithm 4.11. Note, that this heuristic *will* provide a promising column if one exists. We observed that the number of labels for each state (R, i) is very small, typically only one or two. This is a consequence of our insisting on node disjointness of solutions to the pricing problem. For this reason the sophisticated label treatment depicted in Figure 4.5 on page 102 constitutes an overhead, and is not implemented. Similarly, pulling of labels is not an advantage, although both techniques generally have been reported useful by other authors, when node disjointness is not required. We proceed in the treelike manner as indicated in Figure 4.6 on page 107. When *no* column is found for *any* engine we turn off all still active heuristics, if any, thus finally calling an exact pricing algorithm to add final improving columns and to prove optimality. The pricing strategy for a particular engine $e \in \mathcal{E}$ is outlined in Algorithm 5.2 on the next page.

In brief, we employ multiple levels of column generators, ordered by increasing quality which implies increasing computational efforts as well. The most costly generator, *viz.* our exact algorithm is only called when unavoidable. The promising notion of *fast delayed column generation* by OLIVEIRA & FERREIRA (1994) refers to using heuristics instead.

Variants

We experimented with mild deviations from the above. In addition to DANTZIG's pricing rule we alternatively made available both, steepest-edge (option `-s`) and deepest-cut (option `-d`) criteria. When enabled, the respective rule is used to evaluate both, columns in the column pool and labels in Algorithm 4.11. In particular, for steepest-edge pricing we use exact norm calculations which renders this rule computationally far more expensive than deepest-cut for which the exact norm is easy to determine.

Especially when premature termination is enabled in our label correcting algorithm, the order in which patterns are appended to already constructed concatenations may be of importance. We tested two alternatives, chronological order or according to non-increasing dual variable values, and it appears that they are on a par; we use the latter by default.

In Subsection 4.2.6 we presented some more heuristic variants of Algorithm 4.11, almost all of which are tentatively incorporated in our code, c.f. Table 5.1. The options `-e` and `-T` are used

Algorithm 5.2 Overall Pricing Procedure for Engine $e \in \mathcal{E}$ `relevant options`

Update column pool according to current dual variables $\boldsymbol{u}, \boldsymbol{v}$
Add all columns from pool with reduced cost smaller than a threshold `-pa -po`
Delete from pool columns with cost larger than a threshold `-pt`
if columns added from pool **then**
  return
end if
// greedy *maximal dual variable/large remaining slack* heuristic `-Ho`
$R \leftarrow \emptyset$
$T \leftarrow \underline{t}_{e^+} + s_{e^+}$
$C \leftarrow -v_e$
$i \leftarrow e^+$
repeat
  $slack_{r^+} \leftarrow \overline{t}_{r^+} - \max\{\underline{t}_{r^+}, T + t_{ir^+}\} - s_{r^+} \quad \forall r \in \mathcal{R}_e \setminus R$
  $slack_{r^-} \leftarrow \overline{t}_{r^-} - \max\{\underline{t}_{r^-}, \max\{\underline{t}_{r^+}, T + t_{ir^+}\} + s_{r^+} + t_{r^+r^-}\} - s_{r^-} \quad \forall r \in \mathcal{R}_e \setminus R$
  $r^\star \leftarrow \arg\max_{r \in \mathcal{R}_e \setminus R} \{u_r / \max\{\min\{slack_{r^+},\ slack_{r^-}\},\ \varepsilon\} \mid slack_{r^+}, slack_{r^-} \geq 0\}$
  if $r^\star$ defined **then**
    $R \leftarrow R \cup \{r^\star\}$
    $C \leftarrow C + c_{ir^+} + c_{r^+r^-} - u_r$
    $T \leftarrow \max\{\underline{t}_{r^-}, \max\{\underline{t}_{r^+}, T + t_{ir^+}\} + s_{r^+} + t_{r^+r^-}\} + s_{r^-}$
    $i \leftarrow r^-$
  end if
until $r^\star$ undefined, i.e., $slack_{r^+},\ slack_{r^-} < 0 \ \ \forall r \in \mathcal{R}_e \setminus R$
possibly improve C by backtracking, c.f. Algorithm 4.1
if $C < 0$ **then**
  Add column corresponding to C and R to restricted master program
  return
end if
Dual variable based preprocessing via (4.24)
Call Algorithm 4.11 with premature termination enabled `-P`
Store all constructed columns with reduced cost smaller than a threshold in the pool `-pt`

to heuristically restrict the growth of the list of labels; the former enforces shorter concatenations in terms of the number of involved patterns, while the latter excludes concatenations with effective cost larger than a threshold. Also the combination of `-Hi` and `-Hp` may be used to restrict consideration to only a fraction of the *best* requests; dual variable values are taken as quality measure. The tradeoff between short computation times and good column quality is controlled via the premature termination option `-P`. Among the generated columns only the most negative is added to the restricted master, the others are stored to the pool according to option `-pt`. Alternatively, when a negative argument $k < 0$ is given to `-P`, *all* $-k$ negative cost columns are immediately added to the restricted master program.

Multiple Pricing

We try to provide a *good default* which works well on "many" instances. From our experimentation, parts of which are documented in the sequel, we derive the following strategy. Generally, we perform a multiple pricing, more precisely, we cyclically price the engines. We allow for each engine that all profitable columns from the pool with reduced cost below 1.0 are added to the restricted master program. During the first master iterations the heuristic usually furnishes a good column. Later on, more and more often the label correcting algorithm is called which simultaneously refills the column pool. After each such pricing, the restricted master is re-optimized. The motivation for not pricing all engines before re-optimization is to provide the respective pricing problems with the most recent dual information which of course reflects results from prior pricing problems. This is simply a heuristic to avoid generation of too many similar columns.

The decision of multiple *vs.* single pricing can also be determined by the programming environment. CRAMA & OERLEMANS (1994) use the `LINDO` package which does not allow for dynamic modifications of the model in process. Thus, the restricted master program has to be solved from scratch in every iteration, overall a rather costly operation. We experimented with re-optimizing the restricted master after each addition of a column, also when a column is added from the pool. In this case the column pool must contain a negative reduced cost column or it is not used. In effect, this rather slows down the whole process.

In order to avoid cycling in the very final stages of the algorithm it proves useful to define negativity of reduced costs via some threshold; we use less than -10^{-10}.

5.3 Computational Experience for ESP

To put things clear, the code described here is a research prototype, and not a production implementation. It is first and foremost intended for the ease of testing and of evaluating the potential of our approach. All experiments were performed on a 700MHz Pentium III PC with 1GByte core memory running Linux 2.2. We use the `CPLEX` 7.0 callable library to solve the linear and integer programs. Compilation with the GNU C compiler `gcc` is invoked with `-O3` optimization. The program outputs a schedule suggestion for each engine, stating arrival and departure times,

respectively, for each track on its itinerary.

Available Data

We have three data sets available, each of which with its particular characteristics. Basically, an instance is given by the request, engine, and network data, respectively, listed in Table 1.1 on page 11. Requests which are allowed to be served *anywhere* (e.g., breaks) are modeled using a distance of zero from the origin and destination to all other tracks.

VPS Data *Verkehrsbetriebe Peine-Salzgitter GmbH, Germany*, provided us with raw data representing one hundred fulfilled requests from a morning shift of 8.5 hours. All requests originate from operating two blast furnaces at a steel mill, including some additional requests like engineer's breaks. Two engines are primarily responsible for slag transport, and four engines for transportation of molten iron. Only 21 tracks are involved, and distances are relatively short. Creating instance `vps100` from the raw data necessitated considerable manual complementing which has been done with approval by railroad officers. That is, in particular, no time windows were given, but had to be extracted from handwritten data sheets. From the chronologically sorted instance `vps100` we derive a new instance `vps`*nn* by considering the first *nn* requests only. The practical background requires a large fraction of requests to be served full truckload, since much surveillance activity is necessary for safety reasons. A particularity of the data set is that there are many requests compared to the length of the planning horizon.

EKO Data The raw data furnished by *EKO Trans GmbH, Eisenhüttenstadt, Germany*, comes from a steel mill again. This railroad is a smaller one and operates ten engines altogether, each of which is assigned a certain region and/or tasks. In total, we have 820 requests which are generated from a log file of rail car movements on the entire industrial plant during six consecutive days. The requests cover 273 tracks spread all over the plant. No time window information is available for this data set. Therefore, we produce time windows of a parameter controlled width, centered at the actual start-of-service time. This yields instances `eko820a`, `b`, and `c` with time window widths of 200, 250, and 300 minutes, respectively. For our calculations we use groups of 25 requests, such that the time windows span about one half of the respective planning horizon. For many origin destination pairs the distance is estimated. Admittedly, the result cannot be regarded as truly practical data. However, we will use it for some revealing experimentation.

We also generate a set of extremely difficult instances, where time windows are fairly wide, and most distances are *very* short. Then, *many* request sets of almost identical cost are admissible. This results in many columns to be favorably added to the restricted master program, only a tiny fraction of which is actually needed. Instances `eko80xa`, `b`, and `c` each contain requests 1–80 with time window widths of 100, 200, and 300 minutes, respectively.

Sol's Data In his thesis, SOL (1994) randomly generates several *m*-PDPTW instances for computational evaluation of his algorithms (*see* also SAVELSBERGH & SOL 1998). We are kindly

provided with these data sets which we use as raw data for our own experiments. Eight problem classes of ten EUCLIDEAN instances each are given. We have homogeneous fleets, and $|\mathcal{R}|/2$ vehicles are available and admissible for each request. Classes A, B, and DAR have time windows of one hour length, and classes C and D of two hours length. Time windows fulfill $\underline{t}_{r^-} - \underline{t}_{r^+} = t_{r^+r^-}$ for all $r \in \mathcal{R}$. The last time window closes after about eleven hours, however time windows which exceed the planning horizon of 600 minutes are truncated. Vehicle capacity is restricted to accommodate at most three (classes A, C), four (classes B, D), or five requests (classes DAR), respectively. We do not truncate time windows after ten hours, thence we obtain longer planning horizons. From the time window and capacity data we derive for our instances the respective sets of admissible $\mathcal{P}^{1\cup 2}$ patterns. All other information remains intact.

Instance	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{E}\rvert$	$\varnothing\ \lvert\mathcal{E}_r\rvert$	horizon	TW min,∅,max	full!	overl.	embed.
vps40	40	6	4.2	510	0%, 33%, 100%	40%	9%	11%
vps100	100	6	4.0	510	0%, 33%, 100%	42%	8%	9%
					min=∅=max			
eko820a	820	10	2.4	9265	2.2%		51%	52%
eko820b	820	10	2.4	9290	2.7%		52%	53%
eko820c	820	10	2.4	9315	3.2%		52%	53%
eko80xa	80	10	2.6	1028	9.7%		59%	60%
eko80xb	80	10	2.6	1078	18.6%		69%	70%
eko80xc	80	10	2.6	1128	26.6%		78%	79%
				∅horizon	min=∅=max		∅overl.	∅emb.
A30	30	15	15	637.5	9.4%		3.0%	1.1%
B30	30	15	15	643.8	9.3%		4.7%	1.2%
C30	30	15	15	711.1	16.9%		4.2%	2.8%
D30	30	15	15	708.7	16.9%		8.4%	3.8%
A50	50	25	25	649.1	9.2%		4.6%	1.0%
B50	50	25	25	650.4	9.2%		3.5%	1.4%
DAR30	30	15	15	643.7	9.3%		14.3%	5.0%
DAR50	50	25	25	651.5	9.2%		14.1%	4.5%

Table 5.2: Specification of test data. The columns display, respectively, the name of the instance; the number of requests; the number of engines; the average number of admissible engines per request; the length of the planning horizon ($= \max_{i\in\mathcal{N}} \bar{t}_i$), in minutes; the minimal, average, and maximal time window length ($\bar{t}_i - \underline{t}_i - s_i$), respectively, relative to the length of the planning horizon; the fraction of requests which *must* be served full truckload (only vps instances); the fraction of admissible overlapping patterns among all possible overlapping patterns; the same for embedding patterns; in that order. For the classes A, B, C, D, and DAR by SOL (1994) we tabulate the average values for the respective entire classes only.

Discussion of Results

Some observations are generally valid across all instances. At first, all final restricted master programs are integer feasible when default options are used. An intuition for this behavior was already given Section 3.2: All integer solutions are among the basic solutions to the LP relaxation. *See* also our experience in Section 4.3. Secondly, on average solution quality is practically satisfactory, although occasionally significantly above ten percent from optimum. Larger gaps occur e.g., due to small absolute objective function values. We choose the default value to option `-P` not too small, since this enlarges the amount of pooled (and potentially added) columns, which in turn may allow for better integer solutions. Thirdly, with increasing planning horizons, and wider time windows problems become much harder to solve. Our relatively small intermediate linear programs (a) are quickly solved to optimality, (b) never cycle, but (c) could provide more stable dual variables. Moreover, we would confirm the folklore that (d) the *best* columns are generated late or last in the algorithm. This should be kept in mind when an early termination is considered to be incorporated in a branch-and-price environment.

In what regards numerical stability we remark that although all cost coefficients are integers intermediate calculations involving dual variables do have results in the reals. We observe tiny deviations of about 10^{-15} from integral values, and round to the nearest integer. The column headings in the tables displaying our computational results have the following meanings.

Instance	name of the problem instance
LP opt	optimal objective function value of the LP relaxation
IP obj	best feasible integer solution value found by our price-and-branch approach
%gap	relative optimality gap in percent, i.e., $100 \cdot (\text{IP obj} - \lceil \text{LP opt} \rceil) / \lceil \text{LP opt} \rceil$
#eng	number of engines used in our schedule
#opt	minimal feasible number of engines, if known
#pricings	total number of calls to the pricing subproblem; a multiple of $\lvert\mathcal{E}\rvert$
#cols	total number of variables added to the restricted master program
#simplex	total number of simplex iterations performed to re-optimize (RMP$'$)
B&B	CPU time in seconds spent in the final branch-and-bound phase
CPU tot	total computation time in CPU seconds
`-L`	total computation time in CPU seconds, with option `-L` given
mem	peak memory usage of the column generation phase in MByte

VPS The `vps` data set is of special relevance to us since it reflects a typical practical situation. Table 5.3 demonstrates the excellent quality of the lower bound provided by the linear programming relaxation of our set partitioning formulation, and thus the robustness of our price-and-branch approach. Using essentially the default options we obtain integer optimal solutions for a considerable number of requests within a time bound of a few minutes which certainly allows an interactivity with the dispatcher. Due to small absolute cost coefficients we computationally benefit from the `-T` option, and inhibit the generation of labels with cost larger than 30. These experiments also point out how to decompose the planning horizon, if necessary, in order to handle larger instances. The largest problem instance we can integrally solve at once

(however, without certificate of optimality) has 50 requests.

We use these calculations also for an evaluation of our lower bound label elimination criterion (4.19) which we plainly implemented without the presented refinements. As Table 5.3 indicates, for growing problem size, we obtain speedups of orders of magnitude. Using the same amount of time we are enabled to handle additional requests—with considerably smaller memory requirement. The criterion is especially effective in the end of the column generation process, when it is particularly difficult to find negative columns.

In a second experiment, c.f. Table 5.4, we checked different commandline options. We plot in Figure 5.1 the development of the objective function, and the dual variable behavior, respectively, for selected options. We cannot conclude general rules, but trends become visible, and confirm observations we make elsewhere. We observe two significant exceptions regarding computation time. Firstly, option `-P` reveals that requiring a better solution quality of the pricing problem quickly fills the column pool as well, and this in turn enlarges the restricted master program. Most drastically, it is entirely impractical to always run our exact label correcting algorithm until the end. Note, that this remains true when we limit the number of columns to be added from the pool, since the computational bottleneck is the pricing algorithm, c.f. Table 5.7. Second is the effectiveness of label elimination by our lower bound, as mentioned before. Both findings verify that pricing heuristics should be used and/or exact algorithms must be drastically sped up.

We experienced that using no initial heuristic at all, i.e., start with an artificial basis only is competitive to using a greedy heuristic. As discussed earlier, the penalty cost for artificial variables should not be chosen too large. Column elimination is not favorable, especially since this drains chances to obtain integer solutions, and only minor progress is made in terms of the primal objective function per master iteration. Note also that different pricing rules do not differ computationally in these test runs. We remark that the integer solution actually found depends on the setting of options, however, no regularity emerges from Table 5.4. Interestingly, good solutions are obtained also when we limit our search heuristically, e.g., with options `-e` or `-T`. This is important for a later practical employment.

The plots in Figure 5.1 give some complementary information. From the dual variable point of view, turning off the greedy pricing heuristic seems to give a slightly more targeted behavior, while our simple stabilization, using static parameters in (2.18), does not yield significant improvement. When we add *all* promising columns in the early stages of the algorithm (this is implied by option `-P 0`) this only enlarges the restricted master program without forcing any noticeable progress in the objective function.

EKO One purpose of our computational testing is to demonstrate the limitations of our implementation as well. Our experiments with the `eko` data set aim at a performance evaluation with respect to time window width. In practice, we usually have a mix of requests which become available late compared to their required completion time, and of requests which have extremely large time windows, c.f. the `vps` data set. We list results for different time window widths (uniformly given for all requests) in Table 5.5. Note that each group of 25 requests represents approximately one half of a shift. When time windows get wider (and all other data stays the same) the prob-

Instance	LP opt	IP obj	%gap	#eng	#pric.	#cols	#simplex	B&B	CPU tot	-L
vps10	1.000	1.0	**0.00**	5	42	106	59	0.01	0.05	0.10
vps11	1.000	1.0	**0.00**	5	48	87	51	0.00	0.05	0.24
vps12	1.000	1.0	**0.00**	5	48	88	53	0.00	0.05	0.25
vps13	1.000	1.0	**0.00**	5	48	150	93	0.00	0.06	0.31
vps14	9.000	9.0	**0.00**	5	48	99	77	0.00	0.05	0.24
vps15	8.000	8.0	**0.00**	5	60	182	133	0.01	0.10	0.65
vps16	8.000	8.0	**0.00**	5	66	297	233	0.03	0.12	0.55
vps17	8.500	9.0	**0.00**	5	90	432	346	0.07	0.20	1.29
vps18	8.500	9.0	**0.00**	6	78	291	301	0.04	0.14	1.30
vps19	8.000	8.0	**0.00**	6	84	404	347	0.05	0.23	1.61
∅	5.400	5.5	**0.00**	5.2	61.2	213.6	169.3	0.02	0.10	0.65
vps20	8.000	8.0	**0.00**	6	102	934	566	0.33	0.63	2.09
vps21	8.000	8.0	**0.00**	6	72	704	506	0.16	0.44	5.94
vps22	8.000	8.0	**0.00**	6	84	362	369	0.03	0.17	3.76
vps23	8.000	8.0	**0.00**	6	84	471	376	0.10	0.32	9.36
vps24	12.000	12.0	**0.00**	6	90	1683	1819	0.27	2.25	11.88
vps25	8.000	8.0	**0.00**	6	96	1131	676	0.08	1.31	19.60
vps26	8.000	8.0	**0.00**	6	126	1207	1288	0.07	2.25	58.30
vps27	16.000	16.0	**0.00**	6	126	1270	1288	0.09	5.80	65.41
vps28	20.000	20.0	**0.00**	6	144	1598	1508	0.12	11.25	73.18
vps29	19.000	19.0	**0.00**	6	132	1838	1932	0.16	10.33	145.25
∅	11.500	11.5	**0.00**	6.0	105.6	1119.8	1032.8	0.14	3.48	39.47
vps30	19.000	19.0	**0.00**	6	138	2276	2046	0.20	15.32	388.20
vps31	18.000	18.0	**0.00**	6	162	2711	2803	0.31	26.50	512.95
vps32	16.750	17.0	**0.00**	6	162	2645	2819	0.84	25.25	1347.25
vps33	16.750	17.0	**0.00**	6	210	2353	2603	0.89	30.52	15224.63
vps34	16.750	17.0	**0.00**	6	204	2904	3317	1.31	96.29	54707.18
vps35	16.750	17.0	**0.00**	6	198	3001	3461	0.86	58.45	501309.84
vps36	15.000	15.0	**0.00**	6	210	3645	3702	0.88	124.10	†
vps37	22.500	24.0	4.34	6	198	2956	4568	4.41	1862.31	†
vps38	22.500	24.0	4.34	6	192	3144	5170	2.85	1321.49	†
vps39	22.500	27.0	17.39	6	240	3130	4857	16.32	2429.00	†
∅	18.650	19.5	2.61	6.0	191.4	2876.5	3534.6	2.89	598.92	
vps45	≤21.000	21.0		6	324	6275	7329	4.38	8563.40	†
vps50	≤20.000	20.0		6	372	9052	12881	5.84	285304.99	†

Table 5.3: Results for vps instances using default options. For instances vps45 and vps50 option -T 10 was given, i.e., we have *no* guarantee that the LP solution is optimal. The sign † means that every reasonable time bound is exceeded, even for testing purposes (here: one CPU week).

Instance vps40	LP opt	IP obj	%gap	#pric.	#cols	#simpl.	B&B	CPU tot	mem
defaults	25.000	27.0	8.00	258	2648	5464	2.58	2051.58	92.2
-d	25.000	30.0	20.00	270	3621	6715	36.86	2022.20	91.9
-s	25.000	27.0	8.00	282	3363	5883	5.59	2041.23	86.1
-S	25.000	30.0	20.00	384	3275	8815	16.83	3028.01	109.4
-I	25.000	31.0	24.00	378	5115	8742	137.58	8235.09	116.4
-M 100	25.000	31.0	24.00	300	2691	5340	15.71	1854.00	88.7
-M 1000	25.000	27.0	8.00	258	2648	5464	2.58	2045.22	92.2
-M 10000	25.000	27.0	8.00	306	3601	7197	3.77	3859.36	109.6
-P 10	25.000	27.0	8.00	534	17593	6487	79.34	3488.43	101.0
-P 100	25.000	28.0	12.00	240	3184	5543	4.13	1659.29	86.7
-P 1000	25.000	27.0	8.00	162	12656	5137	34.11	4454.93	140.8
-P 10000	25.000	27.0	8.00	150	56525	10556	2925.00	7000.72	172.3
-P 0	25.000	†		138	1325236	5847		52481.12	
-Ho	25.000	27.0	8.00	348	3666	9641	7.36	2705.33	94.3
-pt 1000 -pa 10	25.000	29.0	16.00	252	3386	5174	6.38	1534.04	91.2
-pt 1000 -pa 1	25.000	27.0	8.00	342	3355	7809	5.06	3887.71	102.0
-dc 100	25.000	27.0	8.00	678	4081	9675	15.22	10882.18	114.4
-dc 10	25.000	31.0	24.00	960	2125	9106	16.28	6679.67	134.2
-dc 1	25.000	†		2946	863	15813		9193.35	
-c	18.000	18.0	**0.00**	234	1777	4148	0.20	1575.60	91.6
-f	25.000	28.0	12.00	264	44741	4119	728.77	2399.28	107.4
-Ha	25.000	27.0	8.00	258	2648	5464	2.55	2056.05	92.2
-He	25.000	27.0	8.00	288	2618	6247	9.08	2438.80	85.5
-Hr	25.000	27.0	8.00	324	3420	7418	6.00	2465.82	108.0
-Hc	25.000	30.0	20.00	288	3331	4844	11.63	4409.92	108.7
-Hf	25.000	31.0	24.00	324	3771	7793	20.18	1729.65	89.6
-Hd 50	25.000	32.0	28.00	318	4587	6060	424.09	5165.54	100.0
-Hp .8 -Hi .01	25.000	27.0	8.00	294	3855	6285	52.60	2236.21	92.5
-e 7	25.000	30.0	20.00	318	3218	7249	58.80	3529.96	93.3
-e 6	25.000	30.0	20.00	300	2310	5049	18.77	2745.12	100.4
-e 5	25.000	27.0	8.00	312	3151	5252	28.86	2493.05	99.7
-e 4	25.000	28.0	12.00	312	89798	5102	172.38	2542.95	156.3
-T 15	28.000	29.0		300	2450	5616	4.33	975.39	61.8
-T 10	27.000	27.0		258	2007	4432	0.51	187.48	22.8

Table 5.4: Results for instance vps40: Impact of commandline options. In two cases we do not obtain integer solutions, indicated by the sign †: When always the exact pricing is called (due to memory failure), and when too many columns are eliminated again from the restricted master program. Check Table 5.1 on page 118 for the meaning of the options.

lems get significantly harder to solve. It appears that the "critical" relative time window width (on average) is between 40% and 50% of the planning horizon. Of course, this depends on the duration of the requests as well. More degree of freedom, i.e., longer time windows and comparatively short distances render the application of our approach computationally impractical. It must be pointed out, however, that these results are obtained with default options active. For the computationally most difficult group of requests 676–700 (time window width 300 minutes) we produce an integer solution with objective value of 356.0 (LP optimum is 243.0) in 1,000 CPU seconds, using the `-T 60` option. With sufficient knowledge about the practical situation this option with an appropriate value can always be successfully applied.

Going even further, when there are almost no preferences given with respect to when a request should be served combinatorial explosion strikes, which is catastrophic (not only) from a performance point of view. In practice, such unbound requests would be scheduled in low level workload periods, i.e., assigned to some engine which would otherwise be idle, e.g., when some service is terminated earlier than expected. Our algorithm is not able to handle too large a fraction of such requests as can be deduced from Table 5.6. It appears that the difficulty with the hard instances `eko80x` is not the comparably large fraction of allowed 2-regular patterns, but the enormous amount of feasible paths. Again we note that using option `-T 30` and allowing only 400 CPU seconds for the column generation phase, we obtain an integer solution with 13% gap for the hardest instance.

Sol It was pointed out by DESAULNIERS, DESROSIERS, ERDMANN, SOLOMON & SOUMIS (2000) that there is no publicly available benchmark test bed of PDPTW instances. One main obstacle is that problem specifications diverge considerably among the applications, which limits their mutual comparability. We are in the lucky position of being able to check our results against at least one larger data set from the literature. In Tables 5.8 and 5.10 we list the results for all 80 instances from this set.

On the general PDPTW instances we impose the additional constraint of allowing $\mathcal{P}^{1\cup 2}$-concatenations only. Interestingly enough, the resulting instances are feasible even for the originally shorter planning horizon of 600 minutes. Moreover, for the original instances SOL (1994) gives optimal total route durations. We therefore have a good opportunity to evaluate the impact of our restriction, and calculate the total route duration for our integer solutions as well, c.f. Table 5.9. In a sense, the original situation seems to be restricted in such a way that using $\mathcal{P}^{1\cup 2}$-concatenations is quite competitive, and solutions are quickly computed.

It must be stated explicitly that our objective function also differs from SOL's in that we do not seek a solution with a minimal number of used vehicles, since this is not favored by railroad management (yet). Still, remarkably often the optimal number is used also in our solutions, and only seldom we use one vehicle in excess. We must observe, however, that our engines are allowed to arrive later than at minute 600. The $\mathcal{P}$-concatenations we produce do therefore violate these time windows (and only these). Hence, it happens, that the solution we calculate is better than the optimum stated in SOL (1994), c.f. instance `D30_4` in Table 5.9. For the `DAR50` instances, c.f. Table 5.10, we also use fewer vehicles than are optimal in the original situation. Again, this is a consequence of our enlarged planning horizon.

Requests	width	%	LP opt	IP obj	%gap	#pricings	#cols	#simplex	B&B	CPU tot
1 – 100	200	47	524.146	637.750	16.63	337.5	1301.0	1068.2	2.61	53.30
	250	49	414.958	431.000	5.29	382.5	1571.7	1303.2	1.77	690.05
	300	54	358.930	375.500	5.93	507.5	1778.2	1862.7	2.72	5364.41
101 – 200	200	42	448.794	471.250	4.73	265.0	1324.0	958.2	1.18	29.02
	250	47	391.502	411.750	6.13	360.0	1538.2	1358.7	1.28	406.46
	300	51	363.296	406.750	12.60	422.5	1738.5	1694.2	6.04	4749.79
201 – 300	200	40	630.983	733.000	11.50	205.0	1151.7	990.0	1.24	15.43
	250	45	525.646	560.750	10.34	275.0	1348.2	1324.2	3.10	144.23
	300	49	477.812	563.250	13.33	300.0	1600.5	1667.0	3.12	1892.02
301 – 400	200	43	512.180	551.500	7.75	262.5	1124.7	760.5	0.44	54.91
	250	49	407.790	441.500	7.62	360.0	1561.7	1283.0	1.50	962.21
	300	53	370.219	387.750	4.17	362.5	1540.0	1340.5	1.02	11378.30
401 – 500	200	45	641.833	804.000	19.18	215.0	1080.7	620.5	0.55	3.54
	250	50	429.229	499.500	14.34	232.5	1217.0	841.0	3.47	11.35
	300	55	366.050	441.250	17.63	270.0	1384.7	1033.2	3.66	49.98
501 – 600	200	39	473.078	500.250	6.17	347.5	1600.7	1211.7	0.99	674.23
	250	45	411.723	433.000	6.38	392.5	1574.7	1564.5	4.20	2240.77
	300	49	357.389	379.750	8.77	402.5	1958.2	1741.0	1.55	19176.23
601 – 700	200	48	387.087	400.250	3.90	310.0	1307.2	1261.0	0.49	1760.05
	250	54	327.878	358.250	9.52	372.5	1643.2	1816.7	5.36	30589.96
	300	59	274.975	283.250	2.96	460.0	1999.7	2379.0	16.68	226231.22
701 – 800	200	39	436.679	484.500	9.09	252.5	1129.5	794.2	1.22	30.25
	250	45	352.609	413.250	14.06	305.0	1530.0	1278.7	7.05	115.24
	300	49	306.879	463.750	36.00	325.0	5282.5	1263.0	15.44	337.55

Table 5.5: Results for instances `eko820a`, `b`, and `c`. 32 groups of 25 requests each are formed. We report the average figures for four groups, respectively, i.e., 100 consecutive requests. Under headings ‘width’ and ‘%’ we list the respective time window width in absolute value and relative to the planning horizon, respectively. *See* the text for instructions how to dramatically speed up the computation times by deviating from the default options.

Computationally, we observe that with larger time windows *and* larger vehicle capacity solution times slightly increase and solution quality drops. In contrast to the `vps` instances, it appears that a large fraction of computation time is spent in the final branch-and-bound phase of the algorithm, possibly because here we have larger integrality gaps in absolute values.

Requests	width	%	LP opt	IP obj	%gap	#pric.	#cols	#simplex	B&B	CPU tot
1 – 20	100	19	526.000	526.000	**0.00**	380	1873	669	0.42	4.03
	200	32	483.000	483.000	**0.00**	470	1838	834	0.47	29.72
	300	41	458.000	458.000	**0.00**	500	1586	680	0.33	53.76
21 – 40	100	39	76.000	76.000	**0.00**	540	3612	2051	3.70	2715.21
	200	56	71.000	71.000	**0.00**	730	5968	2569	0.58	15055.72
	300	66	71.000	71.000	**0.00**	600	138559	1752	31.63	12270.77
41 – 60	100	34	87.000	87.000	**0.00**	770	2714	1880	2.26	1981.36
	200	51	87.000	87.000	**0.00**	860	61677	1963	10.93	18815.16
	300	61	87.000	87.000	**0.00**	880	83799	2173	16.59	19440.64
61 – 80	100	48	100.000	100.000	**0.00**	820	486097	4168	54.64	66658.48
	200	65	100.000	†		910	1343598	5813	199.84	274694.39
	300	73		†						

Table 5.6: Results for hard instances `eko80x`. We report results for groups of only 20 requests, and indicate the respective absolute and relative time window width under headings ‘width’ and ‘%’, respectively. We are unable to obtain solutions for the last group, when time windows get wider, c.f. the sign †. For these two problems the pattern graph has a density of 95%. However, for the last instance, premature termination of the column generation phase after 400 CPU seconds leads to an integer solution with objective function value of 113!

Subroutines, tasks	Percentage CPU time
Treatment (expansion) of labels	55.24
Label management	29.91
Calculation of lower bound (4.19)	9.68
Initialization/heuristic	2.25
`CPLEX`	1.49
Memory management	1.33
Column pool management	0.07
Pricing heuristics	0.03
$\sum$	100.00

Table 5.7: Representative run time profile for our price-and-branch code (solution of `vps36`). The computational bottleneck is the pricing problem. Immediate improvements in terms of runtime are to be expected when more elaborate heuristics come into use.

Instance	LP opt	IP obj	%gap	#eng/#opt	#pricings	#cols	#simplex	CPU tot	B&B
A30_1	1786.875	1836.000	2.74	**10/10**	135	1226	1368	2.00	1.04
A30_2	1981.000	1981.000	**0.00**	**11/11**	135	1369	920	0.87	0.03
A30_3	3226.750	3409.000	5.63	14/13	120	1422	837	5.41	4.88
A30_4	1794.000	1819.000	1.39	**11/11**	120	1242	639	1.06	0.49
A30_5	2025.000	2072.000	2.32	**11/11**	90	1744	508	1.45	0.88
A30_6	1837.000	1878.000	2.23	**11/11**	120	1291	741	0.73	0.22
A30_7	1727.500	1734.000	0.34	**10/10**	105	1268	923	0.77	0.14
A30_8	2350.000	2350.000	**0.00**	**11/11**	120	1531	1010	0.64	0.03
A30_9	2501.000	2501.000	**0.00**	**11/11**	120	1380	931	0.68	0.11
A30_10	2594.000	2594.000	**0.00**	**11/11**	120	1365	1295	0.70	0.12
∅	2182.312	2217.400	1.46	11.1/11.0	118.5	1383.8	917.2	1.43	0.79
B30_1	3024.000	3024.000	**0.00**	**12/12**	105	1695	862	0.68	0.11
B30_2	1849.000	1891.000	2.27	**9/9**	135	1561	1048	1.49	0.51
B30_3	2503.000	2503.000	**0.00**	**12/12**	120	1398	953	0.62	0.03
B30_4	2094.000	2094.000	**0.00**	**10/10**	135	1675	1063	1.06	0.05
B30_5	2072.500	2082.000	0.43	**9/9**	120	1378	801	0.91	0.12
B30_6	2100.000	2100.000	**0.00**	**10/10**	120	1251	682	0.63	0.03
B30_7	1980.200	2018.000	1.86	**10/10**	135	1106	1088	0.97	0.21
B30_8	2978.000	2978.000	**0.00**	**13/13**	135	815	999	0.44	0.05
B30_9	2968.000	2968.000	**0.00**	**14/14**	135	1300	1044	0.60	0.02
B30_10	2264.000	2264.000	**0.00**	**11/11**	120	1866	659	0.89	0.15
∅	2383.270	2392.200	0.45	11.0/11.0	126.0	1404.5	919.9	0.82	0.12
C30_1	1490.000	1490.000	**0.00**	**8/8**	120	1471	1198	1.08	0.03
C30_2	1528.000	1528.000	**0.00**	**8/8**	120	1179	1022	1.22	0.02
C30_3	1865.000	1888.000	1.23	**8/8**	150	1131	1660	3.33	0.40
C30_4	1215.100	1276.000	4.93	8/7	135	1174	1538	3.43	1.24
C30_5	1648.000	1648.000	**0.00**	**8/8**	135	1468	1208	1.60	0.05
C30_6	1985.000	2083.000	4.93	**9/9**	120	1611	1411	3.41	2.25
C30_7	1663.667	1675.000	0.66	**8/8**	135	1254	1453	1.81	0.17
C30_8	1830.250	1866.000	1.91	**9/9**	135	1616	1475	2.62	1.30
C30_9	1337.391	1414.000	5.68	**8/8**	105	1495	1512	2.60	1.17
C30_10	1676.833	1692.000	0.89	**8/8**	120	1512	1326	3.05	0.91
∅	1623.924	1656.000	2.02	8.2/8.1	127.5	1391.1	1380.3	2.41	0.75
D30_1	1539.000	1539.000	**0.00**	**9/9**	150	955	1327	2.04	0.13
D30_2	974.000	974.000	**0.00**	**7/7**	165	1597	1675	7.38	0.05
D30_3	1357.500	1393.000	2.57	9/8	135	1195	1296	2.25	0.78
D30_4	1426.659	1494.000	4.69	9/8	150	1037	1368	2.53	1.00
D30_5	1359.452	1405.000	3.30	**8/8**	135	1374	1117	2.16	0.75
D30_6	1615.000	1681.000	4.08	**8/8**	120	1201	1452	3.71	2.53
D30_7	1109.288	1149.000	3.51	9/8	165	1567	1508	3.22	1.41
D30_8	1614.714	1687.000	4.45	8/7	135	1104	1541	2.13	0.54
D30_9	1340.636	1434.000	6.93	**8/8**	135	1597	1641	8.65	6.65
D30_10	1582.200	1719.000	8.59	8/7	135	1219	1859	18.60	17.12
∅	1391.844	1447.500	3.81	8.3/7.8	142.5	1284.6	1478.4	5.26	3.09

Table 5.8: Results for small instances by SOL (1994)

Instance	completion	SOL (1994)	detour	Instance	completion	SOL (1994)	detour
A30_1	4808	4291	1.12	B30_1	6197	4655	1.33
A30_2	4647	4451	1.04	B30_2	4655	4341	1.07
A30_3	6885	5415	1.27	B30_3	5839	4882	1.19
A30_4	5163	5148	1.00	B30_4	5478	4431	1.23
A30_5	5056	4535	1.11	B30_5	5051	4434	1.13
A30_6	5305	4967	1.06	B30_6	5195	4264	1.21
A30_7	4580	4247	1.07	B30_7	5039	4263	1.18
A30_8	5308	5154	1.02	B30_8	6244	5332	1.17
A30_9	5619	4883	1.15	B30_9	6271	5466	1.14
A30_10	5784	5208	1.11	B30_10	5284	4685	1.12
∅	5315.5	4829.9	1.09	∅	5525.3	4675.3	1.17
C30_1	5326	4664	1.14	D30_1	4626	4460	1.03
C30_2	4814	4674	1.02	D30_2	3891	3855	1.00
C30_3	4879	4011	1.21	D30_3	5202	4737	1.09
C30_4	4609	4205	1.09	D30_4	4758	4765	0.99
C30_5	5190	4426	1.17	D30_5	4527	4593	0.98
C30_6	5985	4547	1.31	D30_6	5613	4257	1.31
C30_7	5147	4270	1.20	D30_7	4541	4302	1.05
C30_8	5472	4778	1.14	D30_8	5094	4293	1.18
C30_9	4948	4246	1.16	D30_9	5703	4110	1.38
C30_10	5109	4230	1.20	D30_10	5302	4499	1.17
∅	5147.9	4405.1	1.16	∅	4925.7	4387.1	1.11

Table 5.9: From the schedules we generate we immediately read the alternative objective function value *total route duration*. This value is listed under the heading 'duration,' in comparison to the optimal route duration reported in SOL (1994), and listed under this heading. We truncated time windows after 600 minutes for these runs. Note, that our route duration may be smaller than SOL's when an additional vehicle is used, *see* instance D30_4. Also, our engines need not return to a depot! Therefore, column 'detour,' which gives the ratio of the two objective function values is only a rough estimation of the actual detour.

Instance	LP opt	IP obj	%gap	#eng/#opt	#pricings	#cols	#simplex	CPU tot	B&B
A50_1	2993.000	2993.000	**0.00**	**17/17**	225	2923	2931	7.82	0.49
A50_2	3767.333	3946.000	4.72	**17/17**	200	2499	3999	65.32	60.46
A50_3	3367.000	3367.000	**0.00**	**16/16**	200	2340	2421	5.08	0.18
A50_4	2801.000	2801.000	**0.00**	**14/14**	250	2547	3867	7.52	0.06
A50_5	2554.000	2554.000	**0.00**	**20/20**	225	2870	2664	3.69	0.07
A50_6	2855.000	2855.000	**0.00**	**15/15**	200	3349	5231	9.19	1.47
A50_7	2892.857	2931.000	1.31	**17/17**	225	2257	3085	7.61	2.54
A50_8	2849.200	2872.000	0.77	**15/15**	200	1972	3160	9.71	2.08
A50_9	2690.667	2753.000	2.30	**14/14**	250	2601	3599	16.10	2.91
A50_10	2660.000	2660.000	**0.00**	**15/15**	200	2599	2795	4.72	0.07
∅	2943.005	2973.200	0.91	16.0/16.0	217.5	2595.7	3375.2	13.67	7.03
B50_1	2686.167	2720.000	1.22	**14/14**	225	2570	3982	28.60	20.79
B50_2	3514.500	3608.000	2.64	**17/17**	200	2428	2233	43.87	38.75
B50_3	2036.000	2069.000	1.62	15/13	200	2119	2902	6.94	0.92
B50_4	2675.667	2712.000	1.34	**14/14**	200	2117	3860	6.22	1.59
B50_5	2972.680	3014.000	1.37	**16/16**	200	2038	3550	6.10	2.24
B50_6	3107.333	3115.000	0.22	**15/15**	200	2318	4179	7.74	1.35
B50_7	3087.000	3088.000	0.03	**16/16**	200	2766	4063	9.46	1.88
B50_8	3790.000	3790.000	**0.00**	**18/18**	225	1816	3187	4.38	0.35
B50_9	2499.000	2499.000	**0.00**	**15/15**	250	2243	3983	10.38	0.16
B50_10	2969.833	2971.000	0.03	16/15	300	2820	4064	10.39	0.58
∅	2933.818	2958.600	0.84	15.6/15.3	220.0	2323.5	3600.3	13.40	6.86
DAR30_1	1357.250	1391.000	2.43	**8/8**	120	1384	1156	2.02	0.75
DAR30_2	1932.000	1932.000	**0.00**	**10/10**	120	1103	849	0.70	0.09
DAR30_3	1342.000	1349.000	0.52	11/10	135	1375	714	0.86	0.13
DAR30_4	1159.500	1249.000	7.67	**9/9**	135	1395	1000	2.89	1.75
DAR30_5	1667.400	1672.000	0.23	**8/8**	120	1108	979	0.77	0.17
DAR30_6	2354.000	2418.000	2.71	**11/11**	135	1524	851	31.85	31.21
DAR30_7	1355.500	1377.000	1.54	**8/8**	150	1209	1166	1.22	0.19
DAR30_8	1273.750	1291.000	1.33	10/9	120	1234	734	0.71	0.17
DAR30_9	1856.250	1861.000	0.21	**9/9**	120	1327	1178	1.08	0.18
DAR30_10	1662.500	1667.000	0.24	**8/8**	135	1301	1402	1.50	0.13
∅	1596.015	1620.700	1.68	9.2/9.0	129.0	1296.0	1002.9	4.36	3.47
DAR50_1	1968.000	2069.000	5.13	15/12	250	2354	2514	49.42	44.59
DAR50_2	2555.389	2687.000	5.12	15/13	225	2550	3697	289.67	284.55
DAR50_3	1848.500	1907.000	3.13	12/14	250	2365	3654	25.04	15.74
DAR50_4	2190.091	2229.000	1.73	13/14	250	2393	3262	13.50	5.62
DAR50_5	1803.000	1900.000	5.37	**13/13**	225	2547	3097	46.68	36.95
DAR50_6	1943.929	2001.000	2.93	13/14	225	2311	3533	12.19	2.22
DAR50_7	2275.750	2304.000	1.23	14/13	175	2084	3246	9.72	2.70
DAR50_8	2056.000	2164.000	5.25	13/12	225	2635	4238	80.81	69.95
DAR50_9	3014.464	3141.000	4.17	14/12	200	2158	3739	21.81	14.38
DAR50_10	1799.833	1853.000	2.94	11/13	250	2520	4998	29.95	5.68
∅	2145.495	2225.500	3.70	13.3/13.0	227.5	2391.7	3597.8	57.87	48.23

Table 5.10: Results for larger and less restricted instances by SOL (1994)

Concluding Remarks

The implementation we report upon here developed over time. Without most of the more elaborate strategies, we were barely able to handle about twenty requests. An interesting remark aside is that minimization of the total route duration for instance `vps10` using our mixed integer formulation (ESPMIP) takes about one hour and a half CPU time. More than 130,000 branch-and-bound nodes are explored and more than 2.5 million simplex iterations are spent. When an integer optimal solution is found after approximately one hour the integrality gap still is 25%.

Further extensive testing on practical data is definitively necessary. Our instances do not allow for a comparison of our objective function values with the realized ones, since engine assignments or actual service completion times are not recorded. It goes without saying, that the data we need for our calculations must be electronically available or at least deducible. Currently, however, especially time window information is lacking. This is also due to the fact that our definition of a transportation request is broader than the one used in practice. The size of the instances we are able to solve within acceptable time corresponds to a planning horizon of more than two hours. According to railroad officers it is seldom sensible to plan further ahead because of the increasing uncertainty of future requests.

The use of more elaborate data structures for the label management, and the development of sophisticated heuristics for the pricing problem would certainly bring further speedup. This is not part of our study. Still, we are able to state computational feasibility of our approach; every day practicability is to be established in a next step.

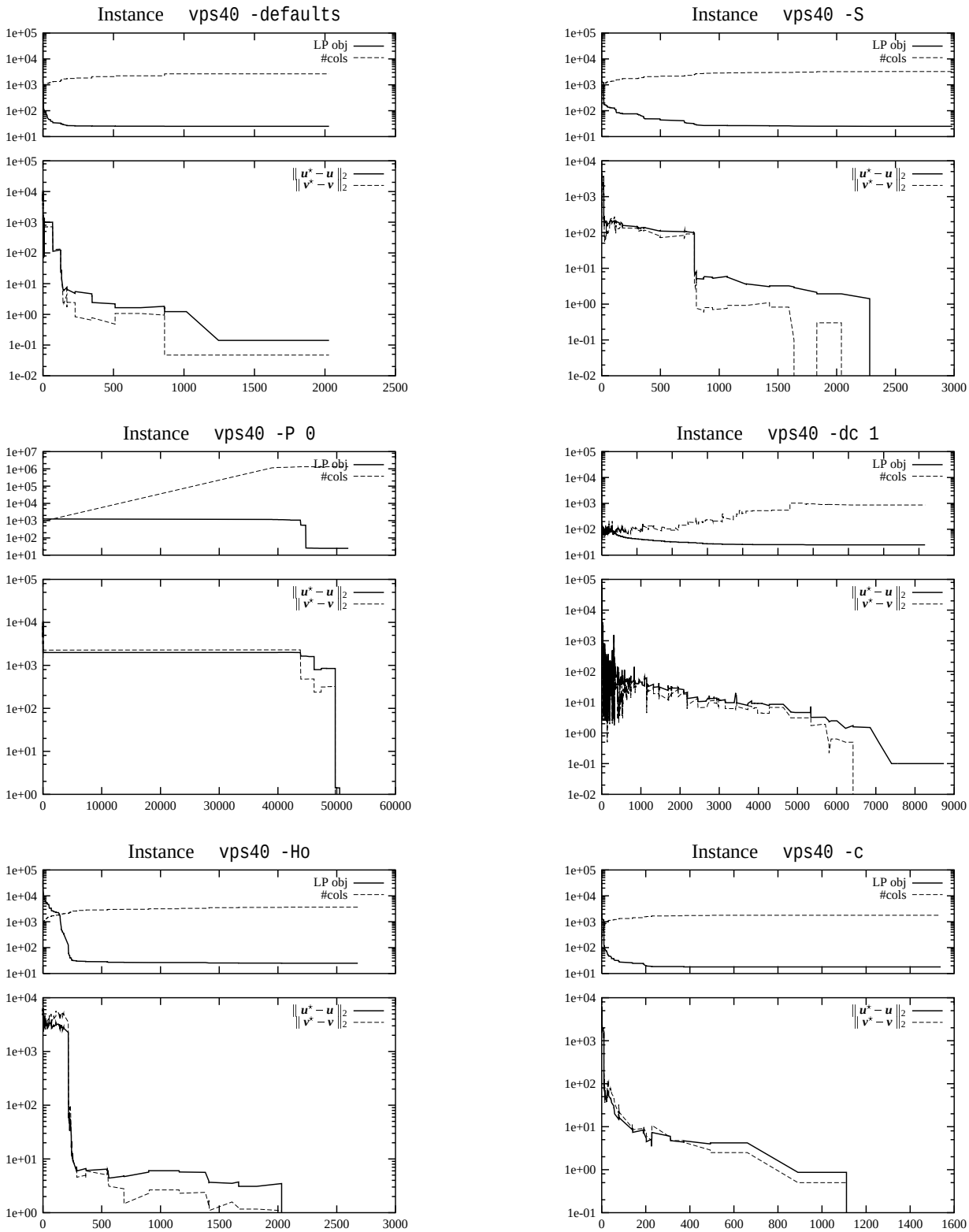

Figure 5.1: For selected commandline options, c.f. Table 5.4, we plot the development of the primal objective function value, the number of variables of the restricted master program, and the convergence of the dual variable vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ towards their respective optimal values.

CHAPTER 6

Combinatorially Restricted Pickup and Delivery Paths

But then you look back at where you've been and a pattern seems to emerge.
And if you project forward from that pattern, then sometimes you can come up with something.
—ROBERT M. PIRSIG, Zen and the Art of Motorcycle Maintenance

Acknowledgment At certain points in writing this chapter I definitely needed controversial discussion, and I wish to thank THOMAS LINDNER, TOM MATSUI, and HANS-HELMUT SCHEEL for their valuable contribution (not only) in this respect.

In Section 1.6 we introduced the general concept of $\mathcal{P}$-concatenations, which allows us to model pickup and delivery paths of a restricted combinatorial nature. For the engine scheduling problem we got by with a very modest choice of a pattern family, motivated by our practical setting. In this chapter we demonstrate that this choice is reasonable from a theoretical point of view as well. Moreover, we comment on the question under which circumstances it pays to adopt our restricted point of view when the original situation is *not* restricted.

Unless otherwise stated, the following assumptions hold throughout this chapter. We consider a single vehicle, the capacity of which is not restrictive, i.e., $\sum_{r\in\mathcal{R}} \ell_r \leq L$. Furthermore, we will not treat individual time windows, although a variant is discussed. Let us denote the number of transportation requests, or customers synonymously, by $|\mathcal{R}| = n$. Recall the request graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ introduced in Section 1.3, c.f. Figure 1.4 on page 11. In this chapter, the engine start and end locations e^+ and e^- are discarded from $\mathcal{N}$. Again, we will make use of the abbreviation $\mathcal{P}^{1\cup\cdots\cup k} := \mathcal{P}^1 \cup \cdots \cup \mathcal{P}^k$. $\mathcal{P}$-concatenations will be assumed request disjoint.

6.1 The Number of Concatenations

Due to the hardness of finding an optimal $\mathcal{P}$-concatenation, c.f. Lemma 1.7, we cannot expect to discover an exact algorithm which is *principally* better than a complete enumeration of feasible solutions, either implicit or explicit. In order to get an impression of the size of the solution space of such algorithms, an interesting question is how many $\mathcal{P}$-concatenations exist, especially in comparison to the general situation of unrestricted pickup and delivery paths.

We investigate consequences of certain *inter* and *intra* pattern structural restrictions. That is, we study particular pattern families as well as the relationship between single patterns in a path. In what concerns the number of $\mathcal{P}$-concatenations, intuitively, *combinatorial explosion* is the more limited the fewer requests appear in different patterns. In this section we prove some results justifying this intuition. Let us first collect some preliminary material.

Definition 6.1 (Number of Concatenations)
Given a family $\mathcal{P}$ of patterns, we denote by $\tau^n_{\mathcal{P}}$ the number of $\mathcal{P}$-concatenations which visit precisely all n customers. In particular, let τ^n be the number of all *pickup and delivery paths.*

Definition 6.2 (Shapes of *k*-Regular Patterns)
Given a fixed numbering $r_1, \ldots, r_n$ of $n \geq k$ requests, the relation $S \subseteq \mathcal{P}^k \times \mathcal{P}^k$ with

$$(P_1, P_2) \in S \iff \text{there exists a permutation on } \{r_1, \ldots, r_n\} \text{ that transforms } P_1 \text{ into } P_2$$

defines an equivalence relation on $\mathcal{P}^k$. The equivalence classes are called shapes*. The number of different equivalence classes, given k, is denoted by σ_k.*

To clarify our definition of shapes recall the notions *overlapping* and *embedding* introduced in Section 1.6. These are the only two shapes of 2-regular patterns, characterized by $\{i^+, j^+, i^-, j^-\}$ and $\{i^+, j^+, j^-, i^-\}$, respectively, for $i \neq j \in \mathcal{R}$. Note, that $\{i^+, i^-, j^+, j^-\}$ is a concatenation of 1-regular patterns, and therefore does not yield a shape of 2-regular patterns. Figure 6.1 shows all ten shapes of 3-regular patterns. From these we construct $\mathcal{P}^3$ by considering for each shape all 3! configurations of three requests out of n.

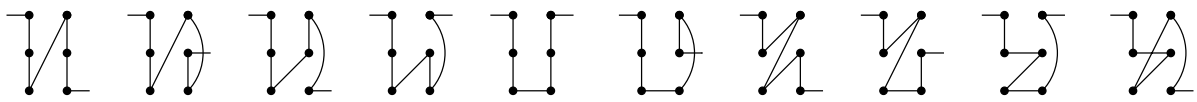

Figure 6.1: All ten shapes of 3-regular patterns. Each left column of nodes represents the origins, each right column the destinations, i.e., each row corresponds to a request. For instance, the leftmost shape is characterized by $\{i^+, j^+, k^+ i^-, j^-, k^-\}$ for $i \neq j \neq k \in \mathcal{R}$. Notice, that by definition no shape contains the concatenation of patterns in $\mathcal{P}^1$ or $\mathcal{P}^2$.

Observation 6.3 *Generally, given $\mathcal{P}$, the number of $\mathcal{P}$-concatenations is*

$$\tau^n_{\mathcal{P}} = \sum_{\substack{P \subseteq \mathcal{P} \\ P \text{ partitions } \mathcal{N}}} |P|! \quad . \tag{6.1}$$

Moreover, we have

$$\tau^n = \frac{(2n)!}{2^n} \quad , \qquad \sigma_k = \frac{\tau^k - \tau^k_{\mathcal{P}^{1\cup\cdots\cup k-1}}}{k!} \quad .$$

The number of all pickup and delivery paths is simply the number of all directed HAMILTONIAN paths on $2n$ nodes, discarding those which do not respect the precedence constraints; k-regular patterns are pickup and delivery paths on k customers not involving patterns in $\mathcal{P}^j$ with $j < k$, and our definition of shapes eliminates symmetry from patterns.

Lemma 6.4 *The number of $\mathcal{P}^{1\cup 2}$-concatenations on $2n$ nodes is*

$$\tau^n_{\mathcal{P}^{1\cup 2}} \;=\; n! \cdot \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \frac{2^i}{i!} \cdot \prod_{j=i}^{2i-1} (n-j) \quad . \tag{6.2}$$

Proof. Each summand represents the number of different combinations of patterns for a fixed number $0 \le i \le \lfloor \frac{n}{2} \rfloor$ of 2-regular patterns involved in the concatenation. We assume that the empty product evaluates to one. For a fixed i there are $(n-2i)$ 1-regular patterns, and therefore $(n-2i+1)$ possibilities to position the first 2-regular pattern. For the second 2-regular pattern there are $(n-2i+2)$ possible positions, and finally $(n-2i+i)$ possibilities to place the last. This yields the stated product. Symmetry in this counting is accounted for by dividing by the number $i!$ of permutations of the 2-regular patterns. Multiplication by $\sigma^i_2 = 2^i$ considers the fact that there are two different 2-regular patterns, i.e., shapes, for a given ordered pair of requests. Having thus calculated the number of configurations of 1-regular and 2-regular patterns, multiplication by $n!$ assigns requests to each particular pattern. This gives (6.2). □

Remark. Using the definition of binomial coefficients $\binom{n}{k} = \frac{n!}{(n-k)!\cdot k!}$ we may rewrite (6.2) as

$$\tau^n_{\mathcal{P}^{1\cup 2}} \;=\; n! \cdot \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \cdot \binom{n-i}{i} \quad . \tag{6.3}$$

An interpretation of (6.3) is that the binomial coefficient gives the number of possibilities to arrange $n-2i$ 1-regular and i 2-regular patterns (neglecting their shapes): For i fixed requests we only have to decide with which of the remaining $n-i$ requests 2-regular patterns will be built.

Although $\tau^n_{\mathcal{P}^{1\cup 2}}$ is exponential in n, its contribution to the total number of possible pickup and delivery paths is negligible. More precisely, we have

Lemma 6.5 $\lim_{n\to\infty} \tau^n_{\mathcal{P}^{1\cup 2}} / \tau^n = 0$.

Proof.

$$\frac{\tau^n_{\mathcal{P}^{1\cup 2}}}{\tau^n} = \frac{n!}{(2n)!}\cdot\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\frac{2^{n+i}}{i!}\cdot\prod_{j=i}^{2i-1}(n-j) \leq \frac{n!}{(2n)!}\cdot n\cdot 2^{1.5n}\cdot n!$$

$$\approx \frac{\left(\frac{n}{\mathrm{e}}\right)^{2n}\cdot 2\pi n}{\left(\frac{2n}{\mathrm{e}}\right)^{2n}\cdot\sqrt{4\pi n}}\cdot n\cdot 2^{1.5n} = \frac{n^{2n}\cdot\sqrt{\pi n}}{(2n)^{2n}}\cdot n\cdot 2^{1.5n} = \frac{n\sqrt{\pi n}}{2^{0.5n}} \longrightarrow 0$$

for $n\to\infty$. The factorials are approximated using STIRLING's formula, where the additive error term $O(n^{-1})$ is super compensated by the overestimating upper bound, and therefore is negligible. □

Lemma 6.6 *The number of $\mathcal{P}^{1\cup\cdots\cup k}$-concatenations on $2n$ nodes is*

$$\tau^n_{\mathcal{P}^{1\cup\cdots\cup k}} \;=\; n!\cdot\sum_{i=0}^{\lfloor\frac{n}{k}\rfloor}\frac{\sigma^i_k}{i!}\cdot\prod_{j=(k-1)i}^{ki-1}(n-j)\cdot\frac{\tau^{n-ki}_{\mathcal{P}^{1\cup\cdots\cup k-1}}}{(n-ki)!}\quad .$$

Proof. The argumentation goes along the lines of the proof of Lemma 6.4. The summation runs over the number $0\leq i\leq\lfloor\frac{n}{k}\rfloor$ of involved k-regular patterns. Again, the different shapes of these are respected by multiplication with σ^i_k. The product counts the number of positions to place i k-regular patterns in between all other patterns involved in the respective concatenation. Hence, we have to account for the number of possible configurations to be built of patterns in $\mathcal{P}^{1\cup\cdots\cup k-1}$ on the remaining $n-ki$ customers not contained in the k-regular patterns. Note, that this is not the respective number of $\mathcal{P}^{1\cup\cdots\cup k-1}$-concatenations, hence we have to adjust $\tau^{n-ki}_{\mathcal{P}^{1\cup\cdots\cup k-1}}$ by division of $(n-ki)!$, thus eliminating symmetry. □

Already for very small n the ratio in Lemma 6.5 is almost zero, c.f. Figure 6.2 on page 145. In a sense, its rate of decrease measures the restriction we impose on pickup and delivery paths. However, with respect to comparing the size of the solution space of an enumeration algorithm, Lemma 6.5 is not fair. We therefore choose to restrict the capacity of the otherwise unrestricted vehicle so as to allow at most two requests loaded simultaneously. Note, that this condition is weaker than requiring $\mathcal{P}^{1\cup 2}$-concatenations, since it includes $i^+,i^+_1,i^-_1,i^+_2,i^-_2,\ldots,i^+_k,i^-_k,i^-$ with $i,i_1,\ldots,i_k\in\mathcal{R}$, $k>1$ as a feasible path segment. For the reason of a simplified presentation we assume $\ell_r=1$ for all $r\in\mathcal{R}$ and $L=2$. Let $\tau^n_{L=2}$ denote the number of pickup and delivery paths feasible for this situation.

Lemma 6.7 $\tau^n_{L=2}=3^{n-1}n!$.

Proof. We prove this result by induction. For $n=1$ we have $\tau^1_{L=2}=3^0\cdot 1!=1$ which is clearly true. Now let $\tau^n_{L=2}=3^{n-1}n!$ hold for an arbitrary but fixed $n\in\mathbb{N}$. We will iteratively construct a path respecting $L=2$. For each stage, given that $i\in\{0,1,2\}$ items are in the vehicle, and

k requests are not yet completed, i.e., k destination nodes are not yet visited, denote by τ_i^k the number of different possibilities to complement the current path from the current node. We seek $\tau_{L=2}^{n+1} = \tau_0^{n+1}$: In the beginning no request is completed and the vehicle is empty.

When the vehicle is empty and k requests are not completed, we have k possibilities to immediately continue the path, i.e., $\tau_0^k = k \cdot \tau_1^k$. When one item is in the vehicle, we can immediately deliver the item, reducing k by one and making the vehicle empty, or we can pick up another item, for which we have $k-1$ possibilities. In the latter case, two items are in the vehicle, thus we must deliver precisely one of them. Then, one item remains in the vehicle, and k is reduced by one, i.e., $\tau_2^k = 2 \cdot \tau_1^{k-1}$ for any k. More formally, we have

$$\tau_0^{n+1} = (n+1) \cdot \tau_1^{n+1} = (n+1) \cdot (\tau_0^n + n \cdot \tau_2^{n+1}) = \\ (n+1) \cdot (\tau_0^n + 2n \cdot \tau_1^n) = (n+1) \cdot (\tau_0^n + 2n \cdot \frac{\tau_0^n}{n}) = 3 \cdot (n+1) \cdot \tau_0^n \ .$$

Hence, by the induction hypothesis and $\tau_0^n = \tau_{L=2}^n$ we have $\tau_{L=2}^{n+1} = 3^n(n+1)!$ as claimed. □

Lemma 6.8 $\lim_{n\to\infty} \tau_{\mathcal{P}^{1\cup 2}}^n / \tau_{L=2}^n = 0$.

Proof. We inductively prove that $\tau_{\mathcal{P}^{1\cup 2}}^n / \tau_{L=2}^n \leq 1/n$ for all $5 \leq n \in \mathbb{N}$ from which the result follows. Using (6.3) we rewrite the claim as

$$n \cdot \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i} \ \leq \ 3^{n-1} \tag{6.4}$$

which is true for $n = 5$. Now, let (6.4) hold for an arbitrary but fixed odd $n \in \mathbb{N}$. For the inductive step consider

$$(n+1) \cdot \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} \ = \ (n+1) \cdot \left[\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n+1-i}{i} + 2^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+1-\lfloor \frac{n+1}{2} \rfloor}{\lfloor \frac{n+1}{2} \rfloor} \right] \tag{6.5}$$

where the rightmost binomial coefficient evaluates to one. We now claim that

$$\text{(a)} \ \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n+1-i}{i} \leq 2 \cdot \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i} \quad \text{and} \quad \text{(b)} \ (n+1) \cdot 2^{\lfloor \frac{n+1}{2} \rfloor} \leq \frac{n-2}{n} \cdot 3^{n-1} \tag{6.6}$$

for all $5 \leq n \in \mathbb{N}$. Then, by application of the induction hypothesis (6.4) in (6.6a), and substituting (6.6) in (6.5) we obtain

$$(n+1) \cdot \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} \ \leq \ 2 \cdot \frac{n+1}{n} \cdot 3^{n-1} + \frac{n-2}{n} \cdot 3^{n-1} \ = \ 3^{n+1-1}$$

as required. The argumentation is analogous for even $n \in \mathbb{N}$. Let us now prove the claims (6.6) in turn. For (6.6a) we inductively show that for all $n \in \mathbb{N}$ it holds that

$$2 \cdot \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i} + (-1)^{n+1} \;=\; \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} \tag{6.7}$$

which immediately implies the claim. One calculates that (6.7) holds for $n = 1$. Given an arbitrary but fixed odd $n \in \mathbb{N}$ we have for $n+1$

$$\begin{aligned}
\sum_{i=0}^{\lfloor \frac{n+2}{2} \rfloor} 2^i \binom{n+2-i}{i} &\overset{(1)}{=} \sum_{i=0}^{\lfloor \frac{n+2}{2} \rfloor} 2^i \left[\binom{n+1-i}{i} + \binom{n+1-i}{i-1} \right] \\
&\overset{(2)}{=} \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} + 2 \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor + 1} 2^{i-1} \binom{n-(i-1)}{i-1} \\
&\overset{(3)}{=} \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} + 2 \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i} \overset{(6.7)}{=} 2 \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} + (-1)^{n+2}
\end{aligned}$$

where (1) follows from the well-known addition theorem for binomial coefficients, odd n is used in (2), and (3) is a substitution of the summation index i by $i+1$, assuming $\binom{n}{-1} = 0$ for $n \in \mathbb{N}$. Again, we omit the analogous part for even $n \in \mathbb{N}$. Finally, rewrite (6.6b) as

$$\frac{n \cdot (n+1)}{n-2} \cdot 2^{\lfloor \frac{n+1}{2} \rfloor} \leq 3^{\lfloor \frac{n+1}{2} \rfloor} \cdot 3^{\lceil \frac{n-3}{2} \rceil} \quad \text{which certainly holds if} \quad \frac{n \cdot (n+1)}{n-2} \leq 3^{\lceil \frac{n-3}{2} \rceil}$$

which is true for $8 \leq n \in \mathbb{N}$; actually, (6.6b) holds for $n = 5, 6, 7$. This completes the proof. □

Remark. The result (6.7) is interesting in its own right. It states that in a modified PASCAL triangle where the i^{th} diagonal is multiplied by the i^{th} power of two, i.e., $\binom{n}{i}$ is replaced by $2^i\binom{n}{i}$, the sum l_n of the diagonal defined by $\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i}$ approximately doubles when n is increased by one. More precisely, these sums behave like the following sequence similar to that of the FIBONACCI numbers, defined by $l_1 = l_2 = 1$ and $l_n = l_{n-1} + 2 \cdot l_{n-2}$ for $2 < n \in \mathbb{N}$.

Lemma 6.9 $\tau^n_{\mathcal{P}^{1\cup 2}} = n! \cdot l_{n+1}$. (6.8)

Proof. Again, we use an inductive argument. Lemma 6.9 holds for $n = 1, 2$. Now suppose the claim be valid for arbitrary but fixed $n-1, n \in \mathbb{N}$ with n even. The proof for odd n is along the same lines and is omitted. Using techniques (1) and (3) from the proof of (6.6a) we compute

$$\begin{aligned}
\tau^{n+1}_{\mathcal{P}^{1\cup 2}} &\overset{(6.3)}{=} (n+1)! \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \binom{n+1-i}{i} \overset{(1)}{=} (n+1)! \sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} 2^i \left[\binom{n-i}{i} + \binom{n-i}{i-1} \right] \\
&\overset{(3)}{=} (n+1)n! \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^i \binom{n-i}{i} + (n+1)n(n-1)! \cdot 2 \sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} 2^i \binom{n-1-i}{i} \\
&\overset{(6.8)}{=} (n+1) \cdot n! l_{n+1} + 2 \cdot (n+1)n \cdot (n-1)! l_n = (n+1)!(l_{n+1} + 2 \cdot l_n) = (n+1)! \cdot l_{n+2}
\end{aligned}$$

where in (3) we additionally factorize $(n+1)!$ and use $\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$, and $\lfloor \frac{n+1}{2} \rfloor - 1 = \lfloor \frac{n-1}{2} \rfloor$. □

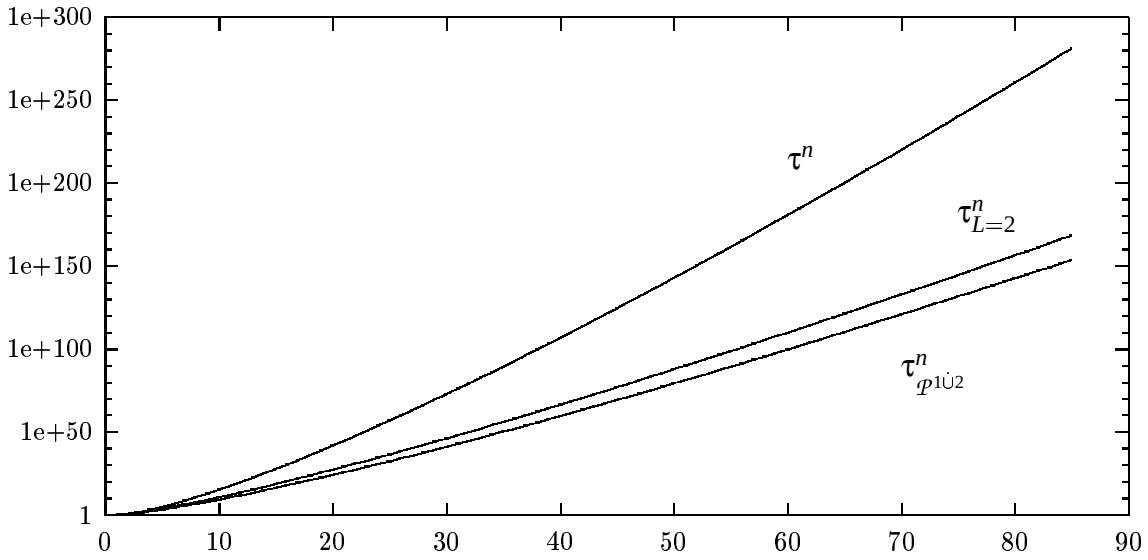

Figure 6.2: Numbers of PDP paths, PDP paths with $L = 2$, and $\mathcal{P}^{1\cup 2}$-concatenations for $n \leq 85$

As Figure 6.2 suggests, compared to a vehicle with capacity two, the restriction to $\mathcal{P}^{1\cup 2}$-concatenations is less severe. Still, asymptotically, their fraction among all pickup and delivery paths with $L = 2$ vanishes. We resume a related topic in the next section. We only remark, that our computational investigations indicate that already the fraction $\tau^n_{\mathcal{P}^{1\cup 2\cup 3}}/\tau^n_{L=2}$ superlinearly diverges with growing n. Let us finally consider pattern families which inherently impose a constraint on the number of patterns to be included in a concatenation.

Lemma 6.10 *Given $\mathcal{P}$, let all partitions of $\mathcal{N}$ consist of exactly $0 \leq k \leq n$ elements of $\mathcal{P}$. Then the number of $\mathcal{P}$-concatenations is bounded from above by*

$$\prod_{i=0}^{k-1} (|\mathcal{P}| - i) \quad .$$

Proof. Using $|P| = k$ in (6.1) we only need to provide an upper bound on the number of partitions of $\mathcal{N}$, which is $\binom{|\mathcal{P}|}{k} = \frac{|\mathcal{P}|!}{(|\mathcal{P}|-k)!\cdot k!}$, and the claim follows. □

For instance, Lemma 6.10 applies to the situation, where $\mathcal{R}$ is partitioned into *morning* and *afternoon* customers, both sets of which are to be visited according to given patterns containing the whole respective set.

6.2 Error Analysis

We will now turn to the investigation of the influence of our restrictions on the quality of solutions in terms of the path length. One would expect that a vehicle handicapped by $\mathcal{P}$-concatenation cannot compete with an unconstrained vehicle. In fact, this is true.

Lemma 6.11 *Let the triangle inequality hold for the cost matrix C. Given the optimal length l of a pickup and delivery path, and the optimal length $l_{\mathcal{P}^{1\cup\cdots\cup k}}$ of a $\mathcal{P}^{1\cup\cdots\cup k}$-concatenation in $\mathcal{G}$ for any fixed $k < n$, then $l_{\mathcal{P}^{1\cup\cdots\cup k}}/l$ is unbounded from above.*

Proof. Let all origins be located at the same physical location O. The same is required for all destinations, located at $D \neq O$. In other words, $c_{i^+j^+} = c_{i^-j^-} = 0$ for all $i \neq j \in \mathcal{R}$, and $c_{1^+1^-} > 0$. The optimal $\mathcal{P}^{1\cup\cdots\cup k}$-concatenation is of length $(2\lceil n/k\rceil - 1)\cdot c_{1^+1^-}$, whereas the shortest pickup and delivery path has length $c_{1^+1^-}$. When k is fixed and n grows, the claim follows. □

Again, this comparison is obviously not fair and corresponds to the result of Lemma 6.5. More meaningfully, we consider an analogue to the situation of Lemma 6.8.

Proposition 6.12 *Let the triangle inequality hold for the cost matrix C. Given the optimal length $l_{L=2}$ of a pickup and delivery path with $L = 2$, and the optimal length $l_{\mathcal{P}^{1\cup 2}}$ of a $\mathcal{P}^{1\cup 2}$-concatenation in $\mathcal{G}$, then*

$$l_{\mathcal{P}^{1\cup 2}} \leq 3 \cdot l_{L=2}$$

and this bound is tight.

Proof. Given the vehicle capacity $L = 2$, let $R_{L=2}$ be an optimal pickup and delivery path in $\mathcal{G}$ of length $l_{L=2}$. In the case that $R_{L=2}$ already is a $\mathcal{P}^{1\cup 2}$-concatenation there is nothing to prove. Otherwise we show how to construct a $\mathcal{P}^{1\cup 2}$-concatenation $R_{\mathcal{P}^{1\cup 2}}$ from $R_{L=2}$. We assume all nodes in $R_{L=2}$ to be located on a straight line in the order they are visited. Among all path structures, every deviation from this solution is penalized to the largest possible extent, because of the validity of the triangle inequality. Since $R_{L=2}$ is not a $\mathcal{P}^{1\cup 2}$-concatenation there exists a 2-regular pattern which is interrupted between its third and fourth node. That is, $R_{L=2}$ contains a sequence $\{i_1^+, i_2^+, i_2^-, R, i_1^-\}$ or $\{i_1^+, i_2^+, i_1^-, R, i_2^-\}$, where $R = \{j_1^+, j_1^-, \ldots, j_k^+, j_k^-\}$ or $R = \{j_1^+, j_1^-, \ldots, j_k^+\}$ for $k \geq 1$. For instance, let $\{i_1^+, i_2^+, i_2^-, \ldots, i_k^+, i_k^-, i_1^-\}$ with $k > 2$ be an ordered subset of $R_{L=2}$. There are several variants to modify this node sequence in order to obtain 1-regular and 2-regular patterns, e.g., $\{i_1^+, i_2^+, i_2^-, i_1^-, i_3^+, i_3^-, \ldots, i_k^+, i_k^-\}$, which is depicted in Figure 6.3 on the next page. In either case, the length of the path segment from i_1^+ to i_1^- is at most tripled in the such constructed $R_{\mathcal{P}^{1\cup 2}}$. By traveling back and forth along the line this way, we can always first complete an interrupted 2-regular pattern before serving the interrupting node sequence R without ever passing a line segment more often than three times. This yields the claimed bound.

When $k = n$ in the above example, and $c_{i_1^+ i_2^+} = c_{i_2^+ i_2^-} = c_{i_n^- i_1^-} = 0$, then the optimal $R_{\mathcal{P}^{1\cup 2}}$ visits requests $i_2, \ldots, i_n$ and then i_1, or vice versa. The length of this concatenation is precisely three times the path length $l_{L=2}$. This completes the proof. □

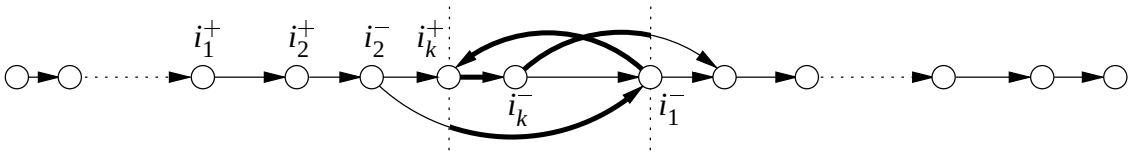

Figure 6.3: Construction of a $\mathcal{P}^{1\cup 2}$-concatenation from a PDP path with $L = 2$

6.3 Polynomially Solvable Cases

From Lemma 1.7 we learn that finding an optimal $\mathcal{P}$-concatenation is $\mathcal{NP}$-complete even for the very simplest pattern families. A natural question in connection with hard optimization problems is to ask for polynomially solvable special cases. In the context of the TSP, two research directions are classical: Special attributes of the cost matrix, and restrictions to special (typically sparse) graph structures. BALAS (1999) considers a third class similar in spirit to our approach, *viz.* constraining the combinatorial variability of a tour. We emphasize the TSP polynomial time special cases—which are numerous—because they immediately give rise to an analogous statement for $\mathcal{P}^1$-concatenation.

Trivially, if the hypothesis of Lemma 6.10 holds for a fixed k, and $|\mathcal{P}|$ is polynomial in n, then there are only $O(|\mathcal{P}|^k)$ concatenations which can be explicitly enumerated. The most general polynomial case which we are able to state, strips almost all of the problem's nature. From our understanding of $\mathcal{P}$-concatenations we conjecture that all extensions to this *core* meant crossing the border to $\mathcal{NP}$-completeness.

Proposition 6.13 *A shortest $\mathcal{P}^{1\cup 2}$-concatenation can be found in polynomial time if the sequence of patterns is irrelevant, i.e., all edges joining two patters have zero weight.*

Proof. Consider the undirected graph with vertex set $V_1 \cup V_2 := \{i \mid i \in \mathcal{R}\} \cup \{i' \mid i \in \mathcal{R}\}$ and edge set $E_1 \cup E_2 \cup E_3 := \{(i,j) \mid i \neq j \in \mathcal{R}\} \cup \{(i',j') \mid i \neq j \in \mathcal{R}\} \cup \{(i,i') \mid i \in \mathcal{R}\}$, i.e., for each request exists a node in V_1 and a copy in V_2 joined by an edge in E_3, and (V_1, E_1) and (V_2, E_2) are complete, c.f. Figure 6.4. Edges $(i,j) \in E_1$ and $(i',j') \in E_2$ are weighted with the cost of a cheapest of the four patterns in $\mathcal{P}^2$ involving requests i and j. Similarly, twice the cost of the full truckload pattern for request i defines the weight for $(i,i') \in E_3$.

Let M be a perfect matching in this graph. The edge sets $M_1 = M \cap (E_1 \cup E_3)$ and $M_2 = M \cap (E_2 \cup E_3)$, respectively, each define a selection of patterns in the obvious way. Furthermore, if M is a minimum weight perfect matching, the edge weight of M_1 equals that of M_2. Otherwise, either $M \cap E_1$ or $M \cap E_2$ would have greater weight, contradicting the optimality of M. Thus, the weight of M is twice the cost of an optimal selection of patterns. Serving these patterns in any order gives an optimal solution to the $\mathcal{P}^{1\cup 2}$-concatenation problem. Polynomial time computability of M by means of EDMONDS' algorithm (*see* e.g., COOK, CUNNINGHAM, PULLEYBLANK & SCHRIJVER 1998) completes the proof. □

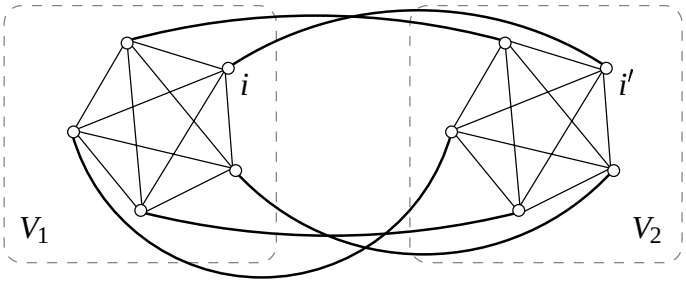

Figure 6.4: Sketch of the undirected graph constructed in the proof of Proposition 6.13

Note, that the (polynomial) matching technique used in the proof does not carry to the analogous $\mathcal{P}^k$ case for $3 \leq k \leq n$, since finding a three dimensional matching is $\mathcal{NP}$-hard.

Restrictions on the Position of a Request Within a Concatenation

We will now develop a non-trivial polynomial time solvable situation for $\mathcal{P}^{1\cup 2}$-concatenation, adopting an interesting idea of BALAS (1999) for the TSP. We follow the lines of his presentation. For a given concatenation, we say that request i *precedes* request j if and only if i^- is visited before j^-, i.e., i is completed not later than j is. Note, that this definition slightly differs from that in Section 3.3. Given an integer $1 \leq k < n$, and a linear ordering $(1,\ldots,n)$ of the set $\mathcal{R}$ of requests, a feasible concatenation will be required to fulfill the following condition:

$$\text{For all } i \neq j \in \{1,\ldots,n\},\ j \geq i+k \Longrightarrow i \text{ precedes } j \ . \tag{6.9}$$

This special precedence constraint requires the *position* of request i within a feasible concatenation to be in the interval $[i-k+1, i+k-1]$. As a generalization, the constant k may be individually chosen for each request. Taking $k = 1$, the ordering $(1,\ldots,n)$ is already optimal. Thus, (6.9) holds for a relatively small k only when the ordering is sufficiently *close* to the optimal one. Figure 6.5 on the facing page depicts a geographical distribution of requests where such an ordering could be known. An optimal concatenation visiting all requests is likely to have a shape close to the curve on the right. We set off by stating BALAS' main result in our terminology.

Theorem 6.14 (BALAS 1999)
Finding an optimal $\mathcal{P}^1$-concatenation with (6.9) can be done in time $O(k^2 2^{k-2} n)$.

An outline of the proof is as follows. The basic idea is to exploit the restricted candidate set for a given position in the concatenation in the well known dynamic programming approach to the TSP. This leads to the construction of a layered graph $G^* = (N^*, A^*)$ with (in our case) $n+2$ layers, containing the node sets N_l^*, $l = 0,\ldots,n+1$, with $N_0^* = \{e^+\}$ and $N_{n+1}^* = \{e^-\}$. Each node in N_l^* represents a full truckload pattern in position l together with the knowledge about which requests were already visited. Two nodes are adjacent if and only if the corresponding requests

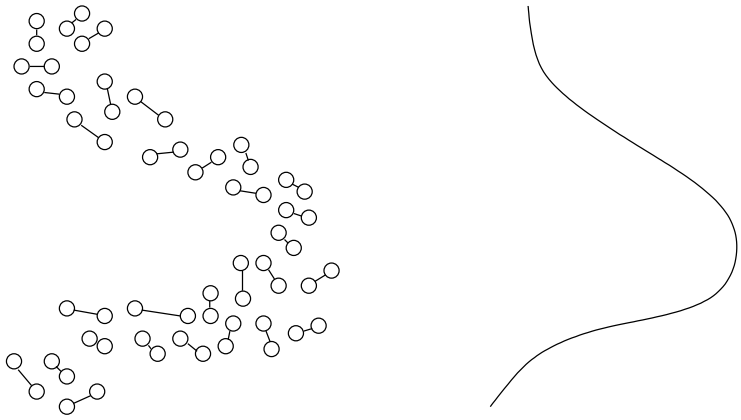

Figure 6.5: One motivation for the precedence constraint (6.9). Requests may be distributed in such a way that intuitively a canonical visiting ordering emerges. Then, with k appropriately chosen so as to reflect the quality of this intuition, we may be able to state a *good* linear ordering.

i and j can be served consecutively according to this knowledge and (6.9). The respective arc is weighted with $c_{i^-j^+} + c_{j^+j^-}$. It can be shown that $|N_l^*| \leq (k+1)2^{k-2}$, $l = 0,\ldots,n+1$, and that the in-degree of every node is bounded by k. This gives an upper bound on the number of arcs in G^*. The claimed result then follows from the one-to-one correspondence between shortest e^+-e^- paths in G^* and optimal $\mathcal{P}^1$-concatenations in $\mathcal{G}$, and the computability of such paths in a layered graph in $O(|A^*|)$. Repeating all the technicalities in this place would be unduly, and we refer to BALAS (1999) for the details. However, this proceeding is important for us inasmuch it suggests an immediate generalization.

Theorem 6.15 *An optimal $\mathcal{P}^{1\cup 2}$-concatenation with (6.9) can be found in time $O(6\cdot k^4 2^{2k-4} n)$.*

Proof. Every e^+-e^- path in G^* represents a feasible sequence of requests to be completed with respect to (6.9). Given $i \in N^*$, denote by $\mathrm{req}(i)$ the associated request. When 1-regular and 2-regular patterns are allowed, for every two adjacent nodes i and j in G^* we have to decide according to which pattern the corresponding requests $\mathrm{req}(i)$ and $\mathrm{req}(j)$ are to be served. Actually, given that $\mathrm{req}(i)$ precedes $\mathrm{req}(j)$, in addition to consecutively serving two full truckload patterns, there are precisely two corresponding $\mathcal{P}^2$-patterns admissible. To represent this, we make four copies $N^*_{l,p,1}$ and $N^*_{l,p,2}$, $p = 1,2$, of the node sets N_l^*, $l = 2,\ldots,n-1$, two copies $N^*_{1,p,1}$, $p = 1,2$, of N_1^*, and two copies $N^*_{n,p,2}$, $p = 1,2$, of N_n^*. The nodes in $N^*_{l,p,1}$ and $N^*_{l,p,2}$, respectively, will represent beginning and ending, respectively, a 2-regular pattern of shape $p \in \{1,2\}$ in the l^{th} position of a concatenation. In each node set $N^*_{l,p,1}$, $p = 1,2$, $l = 1,\ldots,n-1$, we make from each $i \in N_l^*$ precisely $|\delta(i)|$ copies, where $\delta(i)$ denotes the cut induced by $\{i\}$. Denote by $i_{1,\alpha} \in N^*_{l,p,1}$, $\alpha \in \delta(i)$, and $i_2 \in N^*_{l,p,2}$, $p = 1,2$, the respective copies of $i \in N_l^*$, $l = 1,\ldots,n$. All arcs in A^*

remain intact. Additionally, we introduce the following ten arc sets:

$$A^*_{a,p} = \bigcup_{l\in\{1,\ldots,n-1\}} \{(i_{1,\alpha}, j_2) \in N^*_{l,p,1} \times N^*_{l+1,p,2} \mid (i,j) \in A^*, \alpha \in \delta(i)\}, \qquad p = 1,2$$

$$A^*_{b,p} = \bigcup_{l\in\{2,\ldots,n-2\}} \{(i_2, j_{1,\alpha}) \in N^*_{l,p,2} \times N^*_{l+1,p,1} \mid (i,j) \in A^*, \alpha \in \delta(i)\}, \qquad p = 1,2$$

$$A^*_{c,p} = \bigcup_{l\in\{0,\ldots,n-2\}} \{(i, j_{1,\alpha}) \in N^*_{l} \times N^*_{l+1,p,1} \mid (i,j) \in A^*, \alpha \in \delta(i)\}, \qquad p = 1,2$$

$$A^*_{d,p} = \bigcup_{l\in\{2,\ldots,n\}} \{(i_2, j) \in N^*_{l,p,1} \times N^*_{l+1} \mid (i,j) \in A^*\}, \qquad p = 1,2$$

and

$$A^*_{e} = \bigcup_{l\in\{2,\ldots,n-2\}} \{(i_2, j_{1,\alpha}) \in N^*_{l,1,2} \times N^*_{l+1,2,1} \mid (i,j) \in A^*, \alpha \in \delta(i)\}$$

$$A^*_{f} = \bigcup_{l\in\{2,\ldots,n-2\}} \{(i_2, j_{1,\alpha}) \in N^*_{l,2,2} \times N^*_{l+1,1,1} \mid (i,j) \in A^*, \alpha \in \delta(i)\} \ .$$

Let us denote by i^+ and i^-, respectively, the origin and destination location, respectively, of the request associated with node $i \in N^*$. We keep all arc weights w_{ij} for $(i,j) \in A^*$ unchanged, i.e., $w_{ij} = c_{i^-j^+} + c_{j^+j^-}$. We lose no generality in assuming that

$p = 1$ represents the shape $\{j^+, i^+, i^-, j^-\}$ (embedding), and
$p = 2$ represents the shape $\{i^+, j^+, i^-, j^-\}$ (overlapping).

Observe, that each $i_{1,\alpha} \in N^*_{l,p,1}$, $\alpha \in \delta(i)$, $l = 1,\ldots,n-1$, has a unique successor in $N^*_{l+1,p,2}$. By analogy with the above, we denote by $\mathrm{succ}^+(i_{1,\alpha})$ the origin location of the request associated with this successor. This enables us to assign the following weights to the additional arcs:

$$\begin{array}{llll}
(i_{1,\alpha}, j_2) \in A^*_{a,1}: & c_{j^+i^+} + c_{i^+i^-} + c_{i^-j^-} & (i_{1,\alpha}, j_2) \in A^*_{a,2}: & c_{i^+j^+} + c_{j^+i^-} + c_{i^-j^-} \\
(i_2, j_{1,\alpha}) \in A^*_{b,1}: & c_{i^-,\mathrm{succ}^+(j_{1,\alpha})} & (i_2, j_{1,\alpha}) \in A^*_{b,2}: & c_{i^-j^+} \\
(i, j_{1,\alpha}) \in A^*_{c,1}: & c_{i^-,\mathrm{succ}^+(j_{1,\alpha})} & (i, j_{1,\alpha}) \in A^*_{c,2}: & c_{i^-j^+} \\
(i_2, j) \in A^*_{d,1}: & c_{i^-j^+} + c_{j^+j^-} & (i_2, j) \in A^*_{d,2}: & c_{i^-j^+} + c_{j^+j^-}
\end{array}$$

$$\begin{array}{ll}
(i_2, j_{1,\alpha}) \in A^*_{e}: & c_{i^-j^+} \\
(i_2, j_{1,\alpha}) \in A^*_{f}: & c_{i^-,\mathrm{succ}^+(j_{1,\alpha})}
\end{array}$$

Figure 6.6 on the next page gives an overall idea of our network construction. From the definition of our node sets we obtain an upper bound of $(k^2 2^{k-2})^2$ (the square of BALAS' bound) for the

number of arcs entering any $N^*_{l,p,1}$, $p = 1,2$, $l = 1,\ldots,n-1$. This dominates the original upper bound on the number of arcs entering the other node sets. There are six sets with arcs entering some $N^*_{l,p,1}$, *viz.* $A^*_{b,1}$, $A^*_{b,2}$, $A^*_{c,1}$, $A^*_{c,2}$, A^*_e, A^*_f, and the claim follows from Theorem 6.14.

□

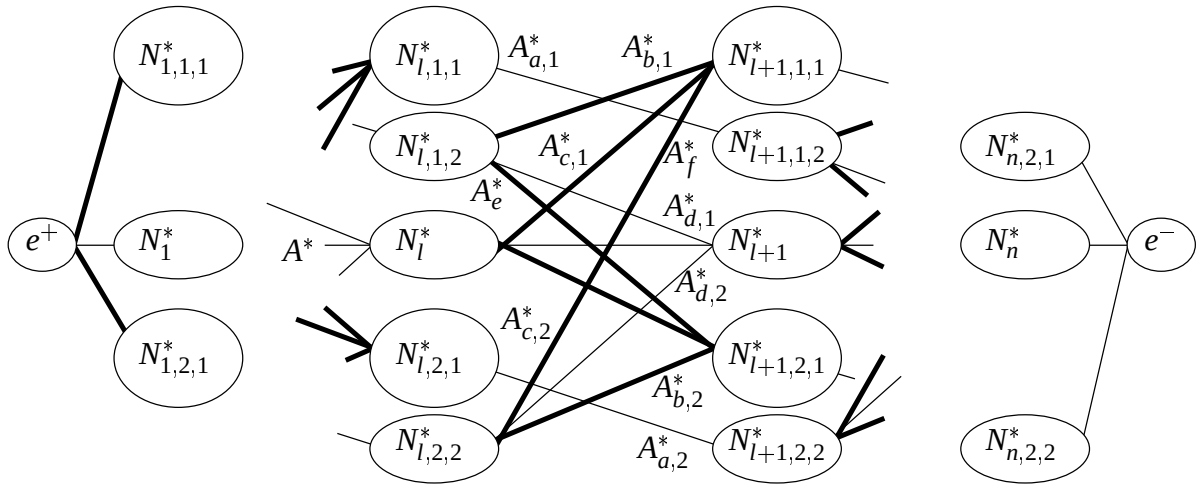

Figure 6.6: Expansion of BALAS' network constructed in the proof of Theorem 6.15. Each line represents the arcs connecting the adjacent node sets. A thin line indicates that the corresponding arc set has cardinality $|A^*|$, whereas bold lines indicate arc sets with cardinality $|A^*|^2$.

It is interesting to see that the embedding patterns cause the costly expansion of BALAS' network. Without the expansion we would not have a unique successor of nodes in $i_{1,\alpha} \in N^*_{l,p,1}$, $\alpha \in \delta(i)$, $l = 1,\ldots,n-1$. Unfortunately, this successor determines which location is to be visited in the embedding pattern first. Thus, we would not have been able to define arc weights e.g., for arcs in $A^*_{b,1}$. Note, however, that for *overlapping* requests the first location to be visited is always known when it is given which request is to be completed first.

Corollary 6.16 *An optimal concatenation of full truckload and overlapping patterns with (6.9) can be found in time* $O(5 \cdot k^2 2^{k-2} n)$.

Proof. Again, we construct an expanded network, starting off from BALAS'. We let $\alpha \equiv 1$ and $p \equiv 2$ in the above construction for the proof of Theorem 6.15. This leads to four arc sets, $A^*_{a,2}$ through $A^*_{d,2}$, each of which has cardinality $|A^*|$. We are able to uniquely define the respective arc weights precisely as above, and the claim follows from Theorem 6.14. □

6.4 Concluding Remarks

By our concept of $\mathcal{P}$-concatenation we are enabled to reduce the search space of an implicit enumeration algorithm for finding an optimal pickup and delivery path, the admissible structure of which must be *a priori* restricted in a certain, well-defined, sense. In fact, all exact solution approaches to the 1-PDP, we are aware of, rely on a dynamic programming scheme. We have proven that $\mathcal{P}^{1\cup 2}$-concatenations give, in a sense, a good representation of all pickup and delivery paths where at most two requests are simultaneously served. The number of the former vanishes compared to that of the latter, but rather surprisingly, we maintain an approximation guarantee. Thus, $\mathcal{P}$-concatenation can validly serve as a heuristic to construct solutions to the PDP. We emphasize, however, that Lemmata 6.5 and 6.8 are asymptotic results, and that the efficiency of an implementation drastically hinges on a relatively small cardinality of the pattern family.

Exact approaches to the multiple-vehicle case usually make use of the single-vehicle case. Therefore, we can hope to having made a contribution to the more general situation as well. We have not considered the practically very relevant occurrence of time windows. At least, Theorems 6.14 and 6.15 and Corollary 6.16 offer polynomial time solubility for the situation that time windows can be transformed to the presented *position windows*.

We discussed pattern families which had a uniform structure. However, much more irregular collections of patterns are conceivable, possibly induced by the introduction of time windows. It would be interesting to investigate special attributes of families which admit further algorithmic benefits, at best, polynomial time solubility. Transferring these results to the TSP might help to identify polynomial special cases also for this prominent combinatorial optimization problem. Furthermore, the requirement that all nodes must be visited could be relaxed.

CHAPTER 7

Reflections

Thank you for a pleasant tale, Wizard. What will you do now?
—TERRY PRATCHETT, The Color of Magic

7.1 Acceptance of Computer Aided Scheduling Tools

Throughout this thesis we kept an emphasis on pointing out the practical background of our research. In this section we discuss various aspects that have to be taken into account when it comes to *installing* a mathematical solution in industry or commerce. We refer to "solution" as an optimization based proposal module in the spirit of a computer aided scheduling tool as introduced in Section 1.4. Our point of view is largely influenced by many discussions with scientists and practitioners in this particular project. We believe that first and foremost the *user's acceptance* of the solution is indispensable in *every* practical setting. Hence, our argumentation mainly deals with achieving this goal and will also demonstrate some of the obstacles we experienced on the way there, a few of which were already sketched by BERNATZKI, BUSSIECK, LINDNER & LÜBBECKE (1998) in a similar situation. *See* also LÜBBECKE & ZIMMERMANN (2001).

The most important insight is that these obstacles are not of mathematical but rather of *human* nature with a psychological background. However, they may have a substantial impact on the choice of the mathematical formulation, on the choice of methods, and on the choice of the implementation. Ignoring them may drain the value of all mathematical and computational work done beforehand. It is therefore advisable to have the support and assistance of a practitioner not only when formulating the problem but also to stimulate his or her feedback at any further stage of the modeling and solution process. This helps to identify and to incorporate additional or modified operational constraints not thought of in the initial problem formulation. Often, as

a starting point, these constraints are not forgotten but left out intentionally in order to quickly provide a first prototype implementation. The purpose of such a model that represents only a simple or simplified scenario is to supply the practitioner with an estimation of the capabilities and potency of a more elaborated model. People are usually not used to see a computer performing a task previously done by an expert human being. We remark, however, that in general a planning process is only partly covered by a model anyway. A reasonable goal should be to depute *routine tasks* to the computer while the planner concentrates on non standard work. This relieves the planner especially during peak workloads and paves the way for a more steady-going decision making process.

The *management* of a firm is interested in the potential financial savings an optimization based planning or decision software tool can offer. However, it has to be stated clearly that it is by no means sufficient to convince the management of the usefulness of such a tool. The reason is that the managerial decision maker is in general not identical to the person who will work with the tool in every day operation. It is therefore even more important to study and carefully evaluate the experience and demands of the *end user*. Simple as it sounds, people *like* or *dislike* novelties in their working environment. It pays to incorporate the features they do like. In the following, we will qualify this claim.

An essential point to make is that a computer still is sometimes seen as an opponent instead of an ally. Even when an existing computer system is to be enhanced by optimization techniques in form of a planning or decision *suggestion* the fear of being *replaced* by the software is often seizable. Objection against the new technology may be a consequence, especially when the underlying methodology is not common knowledge even for experts in the planning environment. The use of mathematical programming is in this sense a *black box* method and solutions are not immediately recognizable by the personnel. Note, that we do not refer to the *mathematical* solution, but to the solution in terms of the practical setting formulated in the language[1] of the practitioner. The structure of such a solution may differ considerably from a manually generated one the planner is used to see. When it does not *feel* right, it *is* not right. Thus, it is helpful to accompany the introduction of the system not only by explaining *how* it works but also by showing examples *that* it works and what practical ideas motivate the differences in the appearance of the solution. A graphical *simulation* clarifying the system's suggestions may be of great help.

One point to particularly focus on is the *quality* of a solution. In mathematical terms, this notion is easily explained via bounds on the objective function value. In practice however, the main point of interest may be different from solutions "guaranteed two percent from optimum." Here, questions of *stability* and *reliability* arise. The former corresponds to the desired property that small changes in the input data shall result in small changes in the output, especially when the input changes frequently. Reliability means that the system is failsafe and provides a solution in *any* circumstance. Imagine that a notification about the infeasibility of the current conditions is of little help for the planner, even if this was the correct answer. The model must be flexible enough

[1]Practitioners need not know, let alone understand mathematical terminology. In contrast, the mathematician *must* know the meaning of technical terms specific to the application area in order to understand the problem. We experienced that the same notion can have different meanings in different communities, and that the same thing can have various names, e.g., a *problem* can be something good or something bad, depending on the point of view.

to cope with such situations, possibly cautioning the operating personal of potential problems, deriving this information from violated constraints. In any case, a good strategy is to compute a heuristic solution in order to have one readily at hand. Time permitting, an improvement of this solution using exact approaches is performed without notice and delivered only when needed.

Of enormous sensitiveness are computerized decisions when people are immediately concerned. This is especially true, when e.g., collective labor agreements have to be respected. The imposed rules are often very complicating in the sense that feasibility can hardly be captured by a rigorous mathematical formulation. BORNDÖRFER & LÖBEL (1999) report in the context of crew scheduling that schedules used in practice—and accepted by the staff council—were *not* feasible with respect to these rules. Also difficult to model are preferences with respect to working with certain colleagues or under certain conditions. Even collecting the required information may pose a severe problem, since this knowledge is understandably scarcely available from the dispatcher who faces his or her loss of influence. To remedy the latter situation, as a generally valid rule, the notion of a computer *suggestion* should be taken literally. The proposal by the system serves as a default only, always subject to the final decision of the human planner. Partial solutions fixed manually must never be overruled by the computer.

We have seen that the trade-off between mathematical rigor and practical needs is often decided in favor of the latter. One may have the impression that the benefits gained from the use of mathematical methods in industry are then compensated. Indeed, shortly after the introduction of a computer aided scheduling system, productivity may decrease significantly and the overall effect may even be negative. Of course, this is only one part of the truth. Once a planner judges the tool useful and reasonable, the default suggestion of the system will be accepted in most cases, not least for the reason of convenience. Building on this basis further components can be added, which makes the introduction of new technologies a dynamic process. We believe that in the long run *remarkable* savings offered by mathematical optimization approaches are definitely worth enduring possibly occurring intermediate embarrassments.

7.2 Contributions

This is a practically oriented thesis in which we investigate the generation of short term schedules for switching engines at industrial in-plant railroads. The problem structure is a multiple-vehicle pickup and delivery problem with time windows. We consider combinatorial restrictions on the sequence of visited locations, introducing the concept of pattern concatenation. Briefly, a pattern is a sequence of locations, feasible as to time windows, precedence, and capacity constraints. We demonstrate that this idea is able to reflect specific requirements in railroad traffic, and we prove that the choice of patterns in our practical situation is also theoretically justified. Within certain limits, the concept finds potential applicability in other practical settings as well.

We show that the construction of schedules of minimal cost with respect to total travel time is $\mathcal{NP}$-complete in the strong sense; the same result holds for finding feasible schedules. Two models for the minimization problem are developed in this thesis. The first is an adaptation of a

mixed integer program known from the vehicle routing literature. It incorporates a distinguishing problem characteristic, *viz.* the requirement that selected patterns imply a partition of the set of requests. From the properties of patterns minor simplifications apply. The model is of polynomial size, but its linear programming relaxation bound is provably poor. Moreover, we demonstrate that this model exhibits a large deal of undesired problem symmetry—a property detrimental to a solution approach by branch-and-bound.

The second formulation is a set partitioning program where a variable corresponds to a feasible schedule for an engine, represented by the subset of visited requests only. The quality of the bound obtained from the linear programming relaxation is proven to be no worse than that of our first model. The number of variables is exponential in the size of the instance, and we propose to solve the linear programming relaxation by column generation. The pricing problem amounts to constructing for a given engine a minimal cost schedule where for each visited request an additional cost of the corresponding dual variable value incurs. We prove that this problem is $\mathcal{NP}$-complete in the strong sense, and develop a label correcting algorithm for its exact solution. Our algorithm differs from existing ones in that it successively appends sequences of nodes to partial solutions instead of single nodes, exploiting our knowledge about the problem structure. Using the current dual information we decide whether further appending is promising. In other words, we make an effort to reduce the label space of the algorithm by rediscovering the use of lower bounds in a dynamic programming algorithm. This idea is new in the pricing context. We also present heuristic methods for the pricing problem, for one of which we prove an approximation guarantee.

Our academic prototype implementation is able to solve the linear programming relaxations of instances with more than 35 requests and six engines in a few minutes which is satisfactory for small railroads, and selected areas of larger ones. This is the largest published ratio of requests to vehicles we are aware of, for which optimal solutions have been furnished. It turns out that for our practical as well as for artificially generated problems the optimal restricted master program always is already integer feasible. The quality of the provided lower bounds, and the obtained solutions, respectively, is excellent.

A theoretical investigation of the underlying problem structure reveals that on the one hand the concept of patterns significantly reduces the number of feasible solutions in contrast to the general pickup and delivery situation. On the other hand we prove that the same concept yields a factor three approximation. Moreover, we are able to state polynomial time soluble special cases of our problem.

This thesis greatly capitalizes on a large body of literature. We expressly aim at providing recent references, if available, on the main topics covered. The extensive bibliography, totaling 190 references, 114 of which no older than 1990, reflects this goal and is intended to contribute in its own right. When there is choice, bias is towards a broad, coherent coverage instead of deep, isolated results. Most notably, Chapter 2 surveys important more recent developments in column generation. A compilation with comparable focus and content was not available formerly.

7.3 Limitations and Perspectives

A fair discussion of our work must exhibit its limits as well. The desired side effect is to open up promising research perspectives. Our computational bottleneck is the exact pricing algorithm. Its proceeding adequately represents the structure of concatenations. However, the practicability of the method hinges on a reasonably small cardinality of the set of admissible patterns. Otherwise, the search space grows prohibitively, mainly due to our insisting on request disjointness of concatenations. A possible remedy, besides dropping the latter requirement, could be to merge pattern concatenation with traditional resource constrained shortest path algorithms. Depending on the structure of admissible patterns, a hybrid algorithm could decide whether to append single nodes or entire patterns. Potentially, such a hybrid required a broader definition of patterns. An investigation of other patterns and properties of more irregular families would be interesting by itself.

Column generation is a powerful technique to solve extremely large (integer) linear programs. On the other hand, it still requires customization and tailoring to the particular application. More generally valid and successful ideas are desired. Several aspects are amenable to improvement in this respect. In choosing an entering column use of, both, primal and dual information, respectively, has been made. However, it is not clear how complementary slackness conditions can be exploited. We cannot give generally valid advice how to best possibly deal with the freedom of choice of dual solutions to be passed to the pricing problem. This includes aspects of progress of the algorithm as well as stability of dual solutions. We deem the concept of additional dual cutting planes very promising. One should definitely strive for a generalization to a more problem independent applicability.

We did not further consider branch-and-price algorithms. From a practical point of view this is well justified by our obtaining always (even optimal) integer solutions. Theoretically, compatibility of branching rules poses a problem. One proposal is to branch by forcing/forbidding certain pattern structures, e.g., particular sequences of nodes. This could be handled easily in the pricing problem. It should be checked whether rules for the PDPTW can be applied. Although our mixed integer program (ESPMIP) is clearly outperformed by our set partitioning formulation, we did not consider to investigate the polyhedral structure of its associated feasible region. Only few results are known in this direction. Strong cutting planes may help to render (ESPMIP) competitive, even though it must be stated that results for related problems are not promising.

Our point of view has been deterministic. From a practical as well as from a theoretical perspective it would be interesting to evaluate stochastic influences of engine availability and appearance of requests. Also, knowledge about the online version of the PDPTW, and in particular of the ESP is scant. A better understanding in this area could contribute to more robust solutions. Moreover, a possible dependency among engines should be taken into consideration. We made first steps towards appropriate model extensions in Section 3.3. Still, we incur eminent difficulties with the pricing problem we currently have no way to efficiently deal with.

This thesis is meant as a reflection of the process of developing a mathematical solution for a practically relevant industrial planning problem. The lack of an appropriate algorithmic solution

in practice stimulated our research. On the other hand, the resulting theoretical achievements, in turn, offer the potential to substantially feed back into daily operation. A work like ours therefore gives an impetus for "both worlds." The growing recognition of mathematics as a *key technology* in industry promises many interesting problems and solutions to come.

finis operis

Bibliography

ADLER, I. & ÜLKÜCÜ, A.(1973), On the number of iterations in Dantzig-Wolfe decomposition, *in* HIMMELBLAU, D.M., (Ed.), 'Decomposition of Large Scale Problems', North-Holland, Amsterdam, pp. 181–187.

AGARWAL, Y., MATHUR, K. & SALKIN, H.M.(1989), 'A set-partitioning-based exact algorithm for the vehicle routing problem', *Networks* **19**, 731–749.

AHUJA, R.K., MAGNANTI, T.L. & ORLIN, J.B.(1993), *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

ALEKSEEV, O.G. & VOLODOS', I.F.(1976), 'Combined use of dynamic programming and branch-and-bound methods in discrete-programming problems', *Automat. Remote Control* **37**, 557–565.

ANBIL, R., FORREST, J.J. & PULLEYBLANK, W.R.(1998), Column generation and the airline crew pairing problem, *in* 'Proceedings of the International Congress of Mathematicians Berlin', Extra Volume ICM 1998 of *Doc. Math. J. DMV*, pp. III 677–686.

APPELGREN, L.H.(1969), 'Column generation algorithm for a ship scheduling problem', *Transportation Sci.* **3**, 53–68.

ASCHEUER, N., KRUMKE, S.O. & RAMBAU, J.(2000), Online dial-a-ride problems: Minimizing the completion time, *in* 'Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science', Volume 1770 of *Lecture Notes in Computer Science*, Springer, pp. 639–650.

ASSAD, A.A.(1980), 'Modelling of rail networks: Toward a routing/makeup model', *Transportation Res. Part B* **14**, 101–114.

AVIS, D. & FUKUDA, K.(1992), 'A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra', *Discr. Comput. Geom.* **8**, 295–313.

BALAS, E. & PADBERG, M.W.(1972), 'On the set-covering problem', *Oper. Res.* **20**, 1152–1161.

BALAS, E. & PADBERG, M.W.(1975), 'On the set-covering problem: II. an algorithm for set partitioning', *Oper. Res.* **23**(1), 74–90.

BALAS, E. & PADBERG, M.W.(1976), 'Set partitioning: A survey', *SIAM Rev.* **18**(4), 710–760.

BALAS, E.(1999), 'New classes of efficiently solvable generalized traveling salesman problems', *Annals of Operations Research* **86**, 529–558.

BARAHONA, F. & ANBIL, R.(1998), On some difficult linear programs coming from set partitioning, IBM Research Report RC 21410 (96674), IBM Research Division.

BARAHONA, F. & ANBIL, R.(2000), 'The volume algorithm: Producing primal solutions with a subgradient method', *Math. Programming* **87**(3), 385–399.

BARAHONA, F. & JENSEN, D.(1998), 'Plant location with minimum inventory', *Math. Programming* **83**, 101–111.

BARNHART, C. & SCHNEUR, R.R.(1996), 'Air network design for express shipment service', *Oper. Res.* **44**(6), 852–863.

BARNHART, C., BOLAND, N.L., CLARKE, L.W., JOHNSON, E.L., NEMHAUSER, G.L. & SHENOI, R.G.(1998), 'Flight string models for aircraft fleeting and routing', *Transportation Sci.* **32**(3), 208–220.

BARNHART, C., HANE, C.A. & VANCE, P.H.(1997), 'Integer multicommodity flow problems', *Lecture Notes in Economics and Mathematical Systems* **450**, 17–31.

BARNHART, C., HANE, C.A. & VANCE, P.H.(2000), 'Using branch-and-price-and-cut to solve origin-destination integer multicommodity network flow problems', *Oper. Res.* **48**(3), 318–326.

BARNHART, C., JOHNSON, E.L., NEMHAUSER, G.L., SAVELSBERGH, M.W.P. & VANCE, P.H.(1998), 'Branch-and-price: Column generation for solving huge integer programs', *Oper. Res.* **46**(3), 316–329.

BEASLEY, J.E. & CHRISTOFIDES, N.(1989), 'An algorithm for the resource constrained shortest path problem', *Networks* **19**, 379–394.

BENDERS, J.F.(1962), 'Partitioning procedures for solving mixed-variables programming problems', *Numer. Math.* **4**, 238–252.

BERNATZKI, K.-P., BUSSIECK, M.R., LINDNER, T. & LÜBBECKE, M.E.(1998), 'Optimal scrap combination for steel production', *OR Spektrum* **20**, 251–258.

BIXBY, R.E. & SALTZMAN, M.J.(1994), 'Recovering an optimal LP basis from an interior point solution', *Oper. Res. Lett.* **15**, 169–178.

BIXBY, R.E., GREGORY, J.W., LUSTIG, I.J., MARSTEN, R.E. & SHANNO, D.F.(1992), 'Very large-scale linear programming: A case study in combining interior point and simplex methods', *Oper. Res.* **40**(5), 885–897.

BIXBY, R.E.(2000*a*), private communication.

BIXBY, R.E.(2000*b*), The linear programming problems, unpublished manuscript.

BOOLER, J.M.P.(1980), 'The solution of a railway locomotive scheduling problem', *J. Opl. Res. Soc.* **31**, 943–948.

BOOLER, J.M.P.(1995), 'A note on the use of lagrangean relaxation in railway scheduling', *J. Opl. Res. Soc.* **46**, 123–127.

BORGWARDT, K.-H.(1982), 'The average number of pivot steps required by the simplex-method is polynomial', *Z. Oper. Res. Ser. A-B* **26**, 157–177.

BORNDÖRFER, R. & LÖBEL, A.(1999), 'Operational planning trends in public transport', Presentation at the Oberwolfach conference "Traffic and Transport Optimization".

BORNDÖRFER, R. & LÖBEL, A.(2001), Scheduling duties by adaptive column generation, ZIB-Report 01-02, Konrad-Zuse-Zentrum für Informationstechnik, Berlin.

BORNDÖRFER, R., GRÖTSCHEL, M., KLOSTERMEIER, F. & KÜTTNER, C.(1999), Telebus Berlin: Vehicle scheduling in a dial-a-ride system, *in* WILSON, N.H.M., (Ed.), 'Computer-Aided Transit Scheduling', Volume 471 of *Lecture Notes in Economics and Mathematical Systems*, Springer, Berlin, pp. 391–422.

BOUCHER, J. & SMEERS, Y.(1986), 'Using column generation techniques for treating dynamic multisectoral models with price-dependent coefficients', *Oper. Res.* **34**(5), 718–725.

BOURJOLLY, J.-M., LAPORTE, G. & MERCURE, H.(1997), 'A combinatorial column generation algorithm for the maximum stable set problem', *Oper. Res. Lett.* **20**(1), 21–29.

BRAMEL, J. & SIMCHI-LEVI, D.(1997), 'On the effectiveness of set covering formulations for the vehicle routing problem with time windows', *Oper. Res.* **45**(2), 295–301.

BRUSCO, M.J. & JACOBS, L.W.(1998), 'Eliminating redundant columns in continous tour scheduling problems', *European J. Oper. Res.* **111**, 518–525.

BUSSIECK, M.R. & LÜBBECKE, M.E.(1998), 'The vertex set of a 0/1-polytope is strongly $\mathcal{P}$-enumerable', *Comput. Geom. Theory Appl.* **11**(2), 103–109.

CHARNES, A. & MILLER, M.H.(1956), 'A model for the optimal programming of railway freight train movements', *Management Sci.* **3**, 74–92.

CHU, H.D., GELMAN, E. & JOHNSON, E.L.(1997), 'Solving large scale crew scheduling problems', *European J. Oper. Res.* **97**, 260–268.

CHVÁTAL, V.(1983), *Linear Programming*, W.H. Freeman and Company, New York.

COOK, W.J., CUNNINGHAM, W.H., PULLEYBLANK, W.R. & SCHRIJVER, A.(1998), *Combinatorial Optimization*, John Wiley & Sons, Chichester.

CORDEAU, J.-F., TOTH, P. & VIGO, D.(1998), 'A survey of optimization models for train routing and scheduling', *Transportation Sci.* **32**(4), 380–404.

CORNUÉJOLS, G. & HARCHE, F.(1993), 'Polyhedral study of the capacitated vehicle routing problem', *Math. Programming* **60**, 21–52.

CRAINIC, T.G. & ROUSSEAU, J.-M.(1987), 'The column generation principle and the airline crew pairing problem', *Infor* **25**, 136–151.

CRAMA, Y. & OERLEMANS, A.G.(1994), 'A column generation approach to job grouping for flexible manufacturing systems', *European J. Oper. Res.* **78**(1), 58–80.

DADUNA, J.R. & WREN, A., (Eds.) (1987), *Computer-Aided Transit Scheduling*, Volume 308 of *Lecture Notes in Economics and Mathematical Systems*, Springer.

DADUNA, J.R., BRANCO, I. & PAIXÃO, J.M.P., (Eds.) (1993), *Computer-Aided Transit Scheduling*, Volume 430 of *Lecture Notes in Economics and Mathematical Systems*, Springer.

DANTZIG, G.B. & WOLFE, P.(1960), 'Decomposition principle for linear programs', *Oper. Res.* **8**, 101–111.

DESAULNIERS, G. & VILLENEUVE, D.(2000), 'The shortest path problem with time windows and linear waiting costs', *Transportation Sci.* **34**(3), 313–323.

DESAULNIERS, G., DESROSIERS, J. & SOLOMON, M.M.(1999), Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems, Les Cahiers du GERAD G-99-36, École des Hautes Études Commerciales, Montréal, Canada.

DESAULNIERS, G., DESROSIERS, J., DUMAS, Y., SOLOMON, M.M. & SOUMIS, F.(1997), 'Daily aircraft routing and scheduling', *Management Sci.* **43**(6), 841–855.

DESAULNIERS, G., DESROSIERS, J., ERDMANN, A., SOLOMON, M.M. & SOUMIS, F.(2000), The VRP with pickup and delivery, *in* TOTH, P. & VIGO, D., (Eds.), 'The Vehicle Routing Problem', SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, chapter 9. Forthcoming.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SOLOMON, M.M., SOUMIS, F. & VILLENEUVE, D.(1998), A unified framework for deterministic time constrained vehicle routing and crew scheduling problems, *in* CRAINIC, T.G. & LAPORTE, G., (Eds.), 'Fleet Management and Logistics', Kluwer, Norwell, MA, pp. 57–93.

DESROCHERS, M. & SOUMIS, F.(1988*a*), 'A generalized permanent labelling algorithm for the shortest path problem with time windows', *Infor* **26**(3), 191–212.

DESROCHERS, M. & SOUMIS, F.(1988*b*), 'A reoptimization algorithm for the shortest path problem with time windows', *European J. Oper. Res.* **35**, 242–254.

DESROCHERS, M. & SOUMIS, F.(1989), 'A column generation approach to urban transit crew scheduling', *Transportation Sci.* **23**, 1–13.

DESROCHERS, M., DESROSIERS, J. & SOLOMON, M.M.(1992), 'A new optimization algorithm for the vehicle routing problem with time windows', *Oper. Res.* **40**(2), 342–354.

DESROCHERS, M., LENSTRA, J.K., SAVELSBERGH, M.W.P. & SOUMIS, F.(1991), Vehicle routing with time windows: Optimization and approximation, *in* GOLDEN & ASSAD (1991), pp. 65–84.

DESROSIERS, J. & DUMAS, Y.(1988), The shortest path problem for the construction of vehicle routes with pickup, delivery and time constraints, *in* 'Advances in Optimization and Control', Volume 302 of *Lecture Notes in Economics and Mathematical Systems*, pp. 144–157.

DESROSIERS, J., DUMAS, Y. & SOUMIS, F.(1986), 'A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows', *American Journal of Mathematical and Management Sciences* **6**, 301–326.

DESROSIERS, J., DUMAS, Y. & SOUMIS, F.(1987), The multiple vehicle dial-a-ride problem, *in* DADUNA & WREN (1987), pp. 15–27.

DESROSIERS, J., DUMAS, Y., SOLOMON, M.M. & SOUMIS, F.(1995), Time constrained routing and scheduling, *in* BALL, M.O., MAGNANTI, T.L., MONMA, C.L. & NEMHAUSER, G.L., (Eds.), 'Network Routing', Vol-

ume 8 of *Handbooks in Operations Research and Management Science*, North-Holland, Amsterdam, pp. 35–139.

DESROSIERS, J., LAPORTE, G., SAUVÉ, M., SOUMIS, F. & TAILLEFER, S.(1988), 'Vehicle routing with full loads', *Comput. Oper. Res.* **15**(3), 219–226.

DESROSIERS, J., PELLETIER, P. & SOUMIS, F.(1983), 'Plus court chemin avec contraintes d'horaires', *RAIRO Rech. Opér.* **17**(4), 357–377. In French.

DESROSIERS, J., SOUMIS, F. & DESROCHERS, M.(1984), 'Routing with time windows by column generation', *Networks* **14**, 545–565.

DESROSIERS, J.(1998), 'Column generation methods', Tutorial given at the INFORMS National Meeting, Montréal, Canada[2].

DESROSIERS, J.(1999*a*), private communication.

DESROSIERS, J.(1999*b*), 'Crew scheduling: Past and future', Presentation at the Oberwolfach conference "Traffic and Transport Optimization".

DESSARPS, P.(2000), Heuristiken für das Pickup-and-Delivery-Problem, Master's thesis, Dept. Mathematical Optimization, Braunschweig University of Technology. In German.

DROR, M.(1994), 'Note on the complexity of the shortest path models for column generation in VRPTW', *Oper. Res.* **42**(5), 977–978.

DU MERLE, O., VILLENEUVE, D., DESROSIERS, J. & HANSEN, P.(1999), 'Stabilized column generation', *Discrete Math.* **194**, 229–237.

DUMAS, Y., DESROSIERS, J. & SOUMIS, F.(1991), 'The pickup and delivery problem with time windows', *European J. Oper. Res.* **54**, 7–22.

DUMAS, Y.(1985), Confection d'itinéraires de véhicules en vue du transport de plusieurs origines à plusieurs destinations, Technical Report 434, Centre de recherche sur les transports, Université de Montréal. In French.

DYER, M.E., RIHA, W.O. & WALKER, J.(1995), 'A hybrid dynamic programming/branch-and-bound algorithm for the multiple-choice knapsack problem', *J. Comput. Appl. Math.* **58**, 43–54.

EASTON, F.F.(1990), 'A dynamic program with fathoming and dynamic upper bounds for the assembly line balancing problem', *Comput. Oper. Res.* **17**(2), 163–175.

EBEN-CHAIME, M., TOVEY, C.A. & AMMONS, J.C.(1996), 'Circuit partitioning via set partitoning and column generation', *Oper. Res.* **44**(1), 65–76.

ERDMANN, A., NOLTE, A., NOLTEMEIER, A. & SCHRADER, R.(1999), Modeling and solving the airline schedule generation problem, Technical Report zpr99-351, ZAIK, University of Cologne, Germany. Submitted to *Math. in Industrial Systems*.

FAGERHOLT, K. & CHRISTIANSEN, M.(2000), 'A combined ship scheduling and allocation problem', *J. Opl. Res. Soc.* **51**(7), 834–842.

FLORIAN, M., BUSHELL, G., FERLAND, J., GUÉRIN, G. & NASTANSKY, L.(1976), 'The engine scheduling problem in a railway network', *Infor* **14**, 121–138.

FORBES, M.A., HOLT, J.N. & WATTS, A.M.(1991), 'Exact solution of locomotive scheduling problems', *J. Opl. Res. Soc.* **42**(10), 825–831.

FORD, L.R. & FULKERSON, D.R.(1958), 'A suggested computation for maximal multicommodity network flows', *Management Sci.* **5**, 97–101.

FORREST, J.J. & GOLDFARB, D.(1992), 'Steepest-edge simplex algorithms for linear programming', *Math. Programming* **57**, 341–374.

FORREST, J.J.(1989), 'Mathematical programming with a library of optimization subroutines', Presented at the ORSA/TIMS Joint National Meeting, New York.

GAMACHE, M., SOUMIS, F., MARQUIS, G. & DESROSIERS, J.(1999), 'A column generation apporach for large-scale aircrew rostering problems', *Oper. Res.* **47**(2), 247–263.

GAREY, M.R. & JOHNSON, D.S.(1979), *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco.

[2]Slides available via `http://www.crt.umontreal.ca/~jacques/Column/index.htm`

GEOFFRION, A.M.(1974), 'Lagrangean relaxation for integer programming', *Math. Programming Stud.* **2**, 82–114.

GILMORE, P.C. & GOMORY, R.E.(1961), 'A linear programming approach to the cutting-stock problem', *Oper. Res.* **9**, 849–859.

GILMORE, P.C. & GOMORY, R.E.(1963), 'A linear programming approach to the cutting stock problem—Part II', *Oper. Res.* **11**, 863–888.

GOFFIN, J.-L. & VIAL, J.-PH.(1999), Convex nondifferentiable optimization: A survey focussed on the analytic center cutting plane method, Technical Report 99.02, Logilab, Université de Genève.

GOFFIN, J.-L., HAURIE, A., VIAL, J.-PH. & ZHU, D.L.(1993), 'Using central prices in the decomposition of linear programs', *European J. Oper. Res.* **64**, 393–409.

GOLDEN, B.L. & ASSAD, A.A., (Eds.) (1991), *Vehicle Routing: Methods and Studies*, Volume 16 of *Studies in Management Science and Systems*, 2nd edition, North-Holland.

GOLDFARB, D. & REID, J.K.(1977), 'A practicable steepest-edge simplex algorithm', *Math. Programming* **12**, 361–371.

HANSEN, P., JAUMARD, B. & POGGI DE ARAGÃO, M.(1998), 'Mixed-integer column generation algorithms and the probabilistic maximum satisfiability problem', *European J. Oper. Res.* **108**, 671–683.

HARRIS, P.M.J.(1973), 'Pivot selection methods of the Devex LP code', *Math. Programming* **5**, 1–28.

HAUPTMEIER, D., KRUMKE, S.O. & RAMBAU, J.(1999), The online dial-a-ride problem under reasonable load, Preprint SC 99-08, Konrad-Zuse-Zentrum für Informationstechnik, Berlin.

HO, J.K.(1984), 'Convergence behavior of decomposition algorithms for linear programs', *Oper. Res. Lett.* **3**(2), 91–94.

HOFFMAN, K.L. & PADBERG, M.W.(1985), 'LP-based combinatorial problem solving', *Annals of Operations Research* **4**, 145–194.

HOLM, S. & TIND, J.(1988), 'A unified approach for price directive decomposition procedures in integer programming', *Discrete Appl. Math.* **20**, 205–219.

HOLMBERG, K. & JÖRNSTEN, K.(1995), 'A simple modification of Dantzig-Wolfe decomposition', *Optimization* **34**(2), 129–145.

HOLMBERG, K., JOBORN, M. & LUNDGREN, J.T.(1998), 'Improved empty freight car distribution', *Transportation Sci.* **32**(2), 163–173.

HURKENS, C.A.J., DE JONG, J.L. & CHEN, Z.(1997), 'A branch-and-price algorithm for solving the cutting strips problem', *Appl. Math., Ser. B (Engl. Ed.)* **12**(2), 215–224.

IBARAKI, T.(1987), *Enumerative Approaches to Combinatorial Optimization*, Volume 10 and 11 of *Annals of Operations Research*, Baltzer.

ILOG INC., CPLEX DIVISION(1997–2000), *Using the CPLEX Callable Library*.

IOACHIM, I., DESROSIERS, J., DUMAS, Y., SOLOMON, M.M. & VILLENEUVE, D.(1995), 'A request clustering algorithm for door-to-door handicapped transportation', *Transportation Sci.* **29**(1), 63–78.

IOACHIM, I., DESROSIERS, J., SOUMIS, F. & BÉLANGER, N.(1999), 'Fleet assignment and routing with schedule synchronization constraints', *European J. Oper. Res.* **119**(1), 75–90.

IOACHIM, I., GÉLINAS, S., SOUMIS, F. & DESROSIERS, J.(1998), 'A dynamic programming algorithm for the shortest path problem with time windows and linear node costs', *Networks* **31**, 193–204.

JAUMARD, B., MEYER, C. & VOVOR, T.(1999), Column/row generation and elimination methods, Les Cahiers du GERAD G-99-34, École des Hautes Études Commerciales, Montréal, Canada.

JOHNSON, E.L., MEHROTRA, A. & NEMHAUSER, G.L.(1993), 'Min-cut clustering', *Math. Programming* **62**, 133–151.

JOHNSON, E.L.(1989), Modelling and strong linear programs for mixed integer programming, *in* WALLACE, S.W., (Ed.), 'Algorithms and Model Formulations in Mathematical Programming', Springer, Berlin, pp. 1–43.

JÜNGER, M. & THIENEL, S.(1998), 'Introduction to ABACUS—a branch-and-cut system', *Oper. Res. Lett.* **22**, 83–95.

KALLEHAUGE, B.(2000), private communication.

KELLEY, J.E. JR.(1961), 'The cutting-plane method for solving convex programs', *J. Soc. Ind. Appl. Math.*

8(4), 703–712.

KIM, K. & NAZARETH, J.L.(1991), 'The decomposition principle and algorithms for linear programming', *Linear Algebra Appl.* **152**, 119–133.

KINDERVATER, G.A.P. & SAVELSBERGH, M.W.P.(1997), Vehicle routing: Handling edge exchanges, *in* AARTS, E. & LENSTRA, J.K., (Eds.), 'Local Search in Combinatorial Optimization', John Wiley & Sons, Chichester, chapter 10, pp. 337–360.

KLEIN, P. & YOUNG, N.(1999), On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms, *in* 'Proceedings of the 7th International IPCO Conference, Graz, Austria', number 1610 *in* 'Lecture Notes in Computer Science', Springer, Berlin, pp. 320–327.

LASDON, L.S.(1970), *Optimization Theory for Large Systems*, Macmillan, London.

LEVENBERG, K.(1944), 'A method for the solution of certain non-linear problems in least squares', *Quart. Appl. Math.* **2**, 164–168.

LINDEROTH, J.T.(1998), *Topics in Parallel Optimization*, PhD thesis, Georgia Institute of Technology, Atlanta, GA.

LÖBEL, A.(1997), *Optimal Vehicle Scheduling in Public Transit*, PhD thesis, Technische Universität Berlin.

LÖBEL, A.(1998), 'Vehicle scheduling in public transit and Lagrangean pricing', *Management Sci.* **44**(12), 1637–1649.

LÜBBECKE, M.E. & ZIMMERMANN, U.T.(2001), Computer aided scheduling of switching engines, Technical report. To appear in 'Mathematics—Key Technology for the Future', Springer, Berlin.

LÜBBECKE, M.E.(2001), Combinatorial restrictions on pickup and delivery paths, Technical report, Dept. Mathematical Optimization, Braunschweig University of Technology. Submitted.

MADSEN, K.(1975), 'An algorithm for minimax solution of overdetermined systems of non-linear equations', *J. Inst. Math. Appl.* **16**, 321–328.

MAGNANTI, T.L., SHAPIRO, J.F. & WAGNER, M.H.(1976), 'Generalized linear programming solves the dual', *Management Sci.* **22**, 1195–1203.

MARCOTTE, O.(1985), 'The cutting stock problem and integer rounding', *Math. Programming* **33**, 82–92.

MARQUARDT, D.W.(1963), 'An algorithm for least-squares estimation of nonlinear parameters', *SIAM J. Appl. Math.* **11**, 431–441.

MARSTEN, R.E. & MORIN, T.L.(1978), 'A hybrid approach to discrete mathematical programming', *Math. Programming* **14**, 21–40.

MARSTEN, R.E., HOGAN, W.W. & BLANKENSHIP, J.W.(1975), 'The BOXSTEP method for large-scale optimization', *Oper. Res.* **23**, 389–405.

MARSTEN, R.E.(1975), 'The use of the boxstep method in discrete optimization', *Math. Programming Stud.* **3**, 127–144.

MEHLHORN, K. & ZIEGELMANN, M.(2000), Resource constrained shortest paths, *in* 'Proceedings of the 8th Annual European Symposium on Algorithms, Saarbrücken, Germany', number 1879 *in* 'Lecture Notes in Computer Science', Springer, pp. 326–337.

MEHROTRA, A. & TRICK, M.A.(1996), 'A column generation approach for graph coloring', *INFORMS J. Comput.* **8**(4), 344–354.

MEHROTRA, A. & TRICK, M.A.(1998), 'Cliques and clustering: A combinatorial approach', *Oper. Res. Lett.* **22**(1), 1–12.

MINGOZZI, A., BIANCO, L. & RICCIARDELLI, S.(1997), 'Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints', *Oper. Res.* **45**(3), 365–377.

MINOUX, M.(1986), *Mathematical Programming*, John Wiley & Sons, Chichester.

MINOUX, M.(1987), 'A class of combinatorial problems with polynomially solvable large scale set covering/partitioning relaxations', *RAIRO Rech. Opér.* **21**, 105–136.

MORIN, T.L. & MARSTEN, R.E.(1976), 'Branch-and-bound strategies for dynamic programming', *Oper. Res.* **24**(4), 611–627.

NAZARETH, J.L.(1984), 'Numerical behaviour of LP algorithms based upon the decomposition principle', *Linear Algebra Appl.* **57**, 181–189.

NAZARETH, J.L.(1987), *Computer Solution of Linear Programs*, Oxford University Press, Oxford.

NEMHAUSER, G.L. & WOLSEY, L.A.(1988), *Integer and Combinatorial Optimization*, John Wiley & Sons, Chichester.

NEMHAUSER, G.L.(1994), 'The age of optimization: Solving large-scale real-world problems', *Oper. Res.* **42**(1), 5–13.

NEMHAUSER, G.L.(1999), private communication.

NEWTON, H.N., BARNHART, C. & VANCE, P.H.(1998), 'Constructing railroad blocking plans to minimize handling costs', *Transportation Sci.* **32**(4), 330–345.

OERTEL, P.(1997), Das Vehicle-Routing-Problem mit Umlademöglichkeit, Master's thesis, Universität zu Köln, Institut für Mathematik. In German.

OLIVEIRA, J.F. & FERREIRA, J.S.(1994), 'A faster variant of the Gilmore and Gomory technique for cutting stock problems', *Belg. J. Oper. Res. Stat. Comput. Sci.* **34**(1), 23–38.

PADBERG, M.W.(1973), 'On the facial structure of set packing polyhedra', *Math. Programming* **5**, 199–215.

PARK, K., KANG, S. & PARK, S.(1996), 'An integer programing approach to the bandwidth packing problem', *Management Sci.* **42**(9), 1277–1291.

PSARAFTIS, H.N.(1980), 'A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem', *Transportation Sci.* **14**, 130–154.

PSARAFTIS, H.N.(1983), 'An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows', *Transportation Sci.* **17**, 351–357.

PSARAFTIS, H.N.(1986), 'Scheduling large-scale advance-request dial-a-ride systems', *American Journal of Mathematical and Management Sciences* **6**, 327–368.

REINGOLD, E.M., NIEVERGELT, J. & DEO, N.(1977), *Combinatorial Algorithms—Theory and Practice*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

RIBEIRO, C.C. & SOUMIS, F.(1994), 'A column generation approach to the multiple-depot vehicle scheduling problem', *Oper. Res.* **42**(1), 41–52.

RIBEIRO, C.C., MINOUX, M. & PENNA, M.C.(1989), 'An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment', *European J. Oper. Res.* **41**, 232–239.

RÖNNQVIST, M., WESTERLUND, A. & CARLSSON, D.(1998), Extraction of logs in forestry using operations research and geographical information systems, Technical report, Division of Optimization, Linköping University, Sweden.

RULAND, K.S. & RODIN, E.Y.(1997), 'The pickup and delivery problem: Faces and branch-and-cut algorithm', *Computers Math. Appl.* **33**(12), 1–13.

RYAN, D.M. & FALKNER, J.C.(1988), 'On the integer properties of scheduling set partitioning problems', *European J. Oper. Res.* **35**, 442–456.

RYAN, D.M. & FOSTER, B.A.(1981), An integer programming approach to scheduling, *in* WREN, A., (Ed.), 'Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling', North-Holland, Amsterdam, pp. 269–280.

SANKARAN, J.K.(1995), 'Column generation applied to linear programs in course registration', *European J. Oper. Res.* **87**(2), 328–342.

SAVELSBERGH, M.W.P. & SOL, M.(1995), 'The general pickup and delivery problem', *Transportation Sci.* **29**(1), 17–29.

SAVELSBERGH, M.W.P. & SOL, M.(1998), 'DRIVE: Dynamic routing of independent vehicles', *Oper. Res.* **46**(4), 474–490.

SAVELSBERGH, M.W.P.(1985), 'Local search in routing problems with time windows', *Annals of Operations Research* **4**, 285–305.

SAVELSBERGH, M.W.P.(1997), 'A branch-and-price algorithm for the generalized assignment problem', *Oper. Res.* **45**(6), 831–841.

SCHRIJVER, A.(1986), *Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester.

SHERALI, H.D. & SMITH, J.C.(2000), Improving discrete model representations via symmetry considerations,

Technical report, Deptartment of Industrial and Systems Engineering, University of Arizona, Tucson.

SMITH, J.C.(2000), private communication.

SOL, M.(1994), *Column Generation Techniques for Pickup and Delivery Problems*, PhD thesis, Eindhoven University of Technology.

SOUMIS, F.(1997), Decomposition and column generation, *in* DELL'AMICO, M., MAFFIOLI, F. & MARTELLO, S., (Eds.), 'Annotated Bibliographies in Combinatorial Optimization', John Wiley & Sons, Chichester, pp. 115–126.

SPIRA, P.M. & PAN, A.(1975), 'On finding and updating spanning trees and shortest paths', *SIAM J. Comput.* **4**, 375–380.

SWEENEY, D.J. & MURPHY, R.A.(1979), 'A method of decomposition for integer programs', *Oper. Res.* **27**, 1128–1141.

SWOVELAND, C.(1974), 'A note on column generation in Dantzig-Wolfe decomposition algorithms', *Math. Programming* **6**, 365–370.

VALÉRIO DE CARVALHO, J.M.(1999), 'Exact solution of bin-packing problems using column generation and branch-and-bound', *Annals of Operations Research* **86**, 629–659.

VALÉRIO DE CARVALHO, J.M.(2000), Using extra dual cuts to accelerate column generation, Technical report, Dept. Produção e Sistemas, Universidade do Minho, Portugal.

VAN DER BRUGGEN, L.J.J., LENSTRA, J.K. & SCHUUR, P.C.(1993), 'Variable-depth search for the single-vehicle pickup and delivery problem with time windows', *Transportation Sci.* **27**(3), 298–311.

VAN ROY, T.J.(1983), 'Cross decomposition for mixed integer programming', *Math. Programming* **25**, 46–63.

VANCE, P.H., ATAMTÜRK, A., BARNHART, C., GELMAN, E., JOHNSON, E.L., KRISHNA, A., MAHIDHARA, D., NEMHAUSER, G.L. & REBELLO, R.(1997), A heuristic branch-and-price approach for the airline crew pairing problem, Technical report.

VANCE, P.H., BARNHART, C., JOHNSON, E.L. & NEMHAUSER, G.L.(1994), 'Solving binary cutting stock problems by column generation and branch-and-bound', *Comput. Optim. Appl.* **3**(2), 111–130.

VANCE, P.H.(1998), 'Branch-and-price algorithms for the one-dimensional cutting stock problem', *Comput. Optim. Appl.* **9**(3), 211–228.

VANDERBECK, F. & WOLSEY, L.A.(1996), 'An exact algorithm for IP column generation', *Oper. Res. Lett.* **19**, 151–159.

VANDERBECK, F.(1994), *Decomposition and Column Generation for Integer Programs*, PhD thesis, Université catholique de Louvain.

VANDERBECK, F.(1995), On integer programming decomposition and ways to enforce integrality in the master, Technical report, Cambridge University. Revised April 1996.

VANDERBECK, F.(1999), 'Computational study of a column generation algorithm for bin packing and cutting stock problems', *Math. Programming* **86**(3), 565–594.

VANDERBECK, F.(2000), 'On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm', *Oper. Res.* **48**(1), 111–128.

WARDROP, A.W.(1987), Scheduling railway motive power, *in* DADUNA & WREN (1987), pp. 250–261.

WEDELIN, D.(1995), 'An algorithm for large scale 0-1 integer programming with application to airline crew scheduling', *Annals of Operations Research* **57**, 283–301.

WENTGES, P.(1997), 'Weighted Dantzig-Wolfe decomposition of linear mixed-integer programming', *Int. Trans. Opl. Res.* **4**(2), 151–162.

WILLIAMS, H.P.(1999), *Model Building in Mathematical Programming*, fourth edition, John Wiley & Sons, Chichester.

WOLSEY, L.A.(1998), *Integer Programming*, John Wiley & Sons, Chichester.

WRIGHT, M.B.(1989), 'Applying stochastic algorithms to a locomotive scheduling problem', *J. Opl. Res. Soc.* **40**(2), 187–192.

ZIARATI, K., SOUMIS, F., DESROSIERS, J. & SOLOMON, M.M.(1999), 'A branch-first, cut second approach for locomotive assignment', *Management Sci.* **45**(8), 1156–1168.

Author and Subject Index

Bold face page numbers are used to indicate pages with important information about the entry, e.g., the *definition* or a detailed explanation, while page numbers in normal type indicate a textual reference.

Notation and Symbols

Different type faces used in the text flag *definitions*, *emphasis*, PERSONS, and ***vectors***.

$\|\cdot\|_2$	EUCLIDEAN norm
$\dot\cup$	disjoint union
$\subseteq$	subset or equal
$\subset$	proper subset
$:=$	equal by definition
$\square$	*quot erat demonstrandum*, end of proof
$[a,b)$	right open interval $a \leq x < b$
2^S	powerset of the set S
$\emptyset$	empty set
$\varnothing$	average
$\mathbf{0}$	vector of all zeros
$\mathbf{1}$	vector of all ones
1-	prefix as in 1-PDP, meaning *single* (vehicle or depot)
$\mathcal{A}$	arc set of a request graph
$\mathcal{A}_\mathcal{P}$	arc set of a pattern graph $\mathcal{G}_\mathcal{P}$
$\mathcal{A}\|_P$	arc set contained in pattern P
$\underline{A}$	constraint matrix of an LP/IP
$\overline{A}$	full constraint matrix of a (restricted) MP, including convexity constraint(s)
A_B	basis matrix corresponding to B
A_J	matrix build of A's columns corresponding to the ordered set $J \subseteq \{1,\ldots,n\}$
A_N	non-basis matrix corresponding to N
$\boldsymbol{a}_q$, $\overline{\boldsymbol{a}}_q$	q^{th} column of A, or $\overline{A}$ respectively
B	index set of basic variables
$\underline{\boldsymbol{b}}$	right hand side corresponding to $\underline{A}$
$\overline{\boldsymbol{b}}$	right hand side corresponding to $\overline{A}$

C_i^k	cost of k^{th} concatenation, lastly visiting node i
C^{inc}	incumbent (currently best reduced cost) value in Algorithm 4.11
(CF)	compact (or original) formulation
$\boldsymbol{c}^\mathsf{T}$	cost vector of an LP/IP
$\bar{\boldsymbol{c}}^\mathsf{T}$	reduced cost vector
c_{ij}	cost incurred when traversing an arc $ij \in \mathcal{A}$
$\bar{c}_{ij}$	reduced cost incurred when traversing an arc $ij \in \mathcal{A}$
$\text{conv}(S)$	convex hull of S
DARP	dial-a-ride problem
(DMP)	dual master program
(DMP′)	linear relaxation of (DMP)
(DRMP)	dual restricted master program
(DRMP′)	linear relaxation of (DRMP)
$\delta(i)$	the cut induced by the node set $\{i\}$, i.e., the set of arcs with tail i
$\boldsymbol{\delta}_R$	incidence vector of set R
$d(P)$	destination node i_k of pattern $P = \{i_1, i_2, \ldots, i_k\}$
$\boldsymbol{\eta}_j$	edge direction corresponding to increasing the j^{th} non-basic variable
$\mathcal{E}$	set of available switching engines
$\mathcal{E}_r$	set of admissible engines for request $r \in \mathcal{R}$
E, E_k	index set of extreme rays
ESP	engine scheduling problem
ESPP	engine scheduling pricing problem
(ESPMIP)	mixed integer formulation of the ESP
(ESPMIP′)	linear relaxation of (ESPMIP)
(ESPSP)	set partitioning formulation of the ESP
(ESPSP′)	linear relaxation of (ESPSP)
$\boldsymbol{e}_j$	j^{th} unit vector
e^+	logical track where engine $e \in \mathcal{E}$ starts its service
e^-	virtual *end of service* location for engine $e \in \mathcal{E}$
$\mathcal{G}$	request graph
$\mathcal{G}_\mathcal{P}$	pattern graph associated with the pattern family $\mathcal{P}$
I	identity matrix
ij, (i, j)	are both used to denote the unique arc from i to j
IP	integer linear program
K	number of components of a block diagonal matrix, and thus, resources
$k \in K$	abbreviates $k = 1, \ldots, K$
$\boldsymbol{\lambda}$	primal variables of a (restricted) master program
$\mathcal{L}(\boldsymbol{u})$	LAGRANGIAN dual function
L_e	tractive effort (i.e., capacity) of engine $e \in \mathcal{E}$
ℓ_r	$(= \ell_{r^+} = -\ell_{r^-})$ size of the load of request $r \in \mathcal{R}$
LB_F	FARLEY's bound on $z^\star_{\text{MP}'}$, *see* Lemma 2.10
LB_L	LASDON's bound on $z^\star_{\text{MP}'}$, *see* Lemma 2.9

LP	linear program
M	"big M" customarily represents a "large number"
MIP	mixed integer linear program
(MP)	master program
(MP′)	linear relaxation of (MP)
m	number of rows of an LP/IP
m-	prefix as in m-TSPTW, meaning *multiple* (vehicle or depot)
$\mathbb{N}$	natural numbers $\{1,2,3,\ldots\}$
$\mathcal{N}$	node set of a request graph
$\mathcal{NP}$	complexity class of non-deterministic polynomial (decision) problems
N	index set of non-basic variables
n	number of columns of an LP/IP; also in Chapter 6: $\lvert\mathcal{R}\rvert = n$
Ω'_e, Ω_e	(sub-)set of admissible request sets for engine $e \in \mathcal{E}$
$o(P)$	origin node i_1 of pattern $P = \{i_1, i_2, \ldots, i_k\}$
$\mathcal{P}$	family of (all) patterns
$\mathcal{P}_e$	subset of patterns admissible for engine $e \in \mathcal{E}$
$\mathcal{P}^1$	family of full truckload patterns
$\mathcal{P}^2$	family of overlapping and embedding patterns
$\mathcal{P}^{1\cup 2}$	family of patterns feasible for the ESP
$\mathcal{P}^k$	k-regular pattern family
$\mathcal{P}^{1\cup\cdots\cup k}$	abbreviates $\mathcal{P}^1 \cup \cdots \cup \mathcal{P}^k$
PDP(TW)	pickup and delivery problem (with time windows)
(PP)	pricing problem
(PP_k)	pricing problem related to k^{th} resource
$\mathbb{Q}$	rational numbers
Q, Q_k	index set of (extreme) points related to all (resp. the k^{th}) resource
Q', Q'_k	respective subsets of Q, Q_k corresponding to the current (RMP)
$\mathbb{R}$	real numbers
$\mathbb{R}_+$, $\mathbb{R}_-$	non-negative, and non-positive real numbers, respectively
$\mathcal{R}$	ground m-set of an SC/SP, especially the request set for ESP
$\mathcal{R}_e$	subset of requests admissible on engine $e \in \mathcal{E}$
$\mathcal{R}_e^+$	$= \{r \in \mathcal{R}_e \mid u_r > 0\}$. $\mathcal{R}_e^-$ is analogously defined
R	subset of $\mathcal{R}$, especially a (feasible) subset of requests
R^{inc}	set of admissible requests associated with the incumbent value C^{inc}
(RMP)	restricted master program
(RMP′)	linear relaxation of (RMP)
r^+	logical origin track of request $r \in \mathcal{R}$
r^-	logical destination track of request $r \in \mathcal{R}$
requests(P)	set of requests visited in pattern $P \in \mathcal{P}$
S, S_k	feasible region of (k^{th}) subproblem
SC	set covering problem
SP	set partitioning problem

σ_k	the number of shapes in $\mathcal{P}^k$
s_{r^+}	service time at origin track of request $r \in \mathcal{R}$
s_{r^-}	service time at destination track of request $r \in \mathcal{R}$
τ^n	number of pickup and delivery paths visiting n customers
$\tau^n_{L=2}$	number of pickup and delivery paths for a vehicle with capacity of two requests
$\tau^n_{\mathcal{P}}$	number of $\mathcal{P}$-concatenations visiting n customers
t_{ij}	traversal time for arc $ij \in \mathcal{A}$
$[\underline{t}_i, \bar{t}_i]$	start-of-service time window for track $i \in \mathcal{N}$
T_i^k	start-of-service time of k^{th} concatenation, lastly visiting node i
TSP(TW)	traveling salesperson problem (with time windows)
$\boldsymbol{u}$	dual variables associated with linking, especially partitioning, constraints
$\boldsymbol{x}$	primal variables of a compact formulation
$\boldsymbol{x}^\mathsf{T}$	transpose of vector $\boldsymbol{x}$
$\boldsymbol{x}^\star$	optimal solution to an LP/IP
VRP(TW)	vehicle routing problem (with time windows)
$\boldsymbol{v}$	dual variables associated with convexity constraints
$\boldsymbol{w}$	dual variables used for various examples
$\mathbb{Z}$	integer numbers
$\mathbb{Z}_+$	non-negative integer numbers
z_P	objective function value of LP/IP P (subscript P omitted if contextually known)
$z_e^\star$	optimum of the pricing problem for engine $e \in \mathcal{E}$
$z_P^\star$	optimal objective function value of LP/IP P

Zusammenfassung

Die vorliegende praktisch orientierte Arbeit beschäftigt sich mit der Einsatzplanung von Rangierlokomotiven bei Industriebahnen. Transportaufträge, charakterisiert durch Start- und Zielgleis, Bedienzeiten nebst Zeitfenstern an beiden Gleisen und Ladungsgröße müssen von einer Menge von technisch und personell unterscheidbaren Loks durchgeführt werden. Nicht jeder Auftrag ist auf jeder Lok fahrbar. Zu gegebenem Zeithorizont zielt die Planung auf einen effektiven Einsatz der Lokomotiven, hier: die Minimierung von Leerfahrten und Wartezeiten. Wir stellen das Problem und seine Verwandtschaft zu *Pickup and Delivery Problemen mit Zeitfenstern* in Kapitel 1 ausführlich dar. Ergänzend hierzu definieren wir (allgemein) mögliche Bediensequenzen, sog. Muster, und führen unsere Aufgabe auf die Verkettung sehr einfacher Muster für jede Lokomotive zurück. Wir zeigen, dass bereits das Auffinden zulässiger solcher Ketten $\mathcal{NP}$-schwer ist und geben Beispiele möglicher Anwendungsgebiete dieses Konzepts nicht nur im Schienenverkehr.

Kapitel 2 vertieft einige ausgewählte Themen zur Spaltengenerierung, d.h. der auf dem Simplexverfahren basierenden Lösung eines linearen Programms (LP), bei dem die Koeffizientenmatrix spaltenweise nur soweit im Verfahren nötig bekannt ist. Diese Technik ermöglicht die Lösung sehr großer LPs und ist die Basis zur Lösung sehr großer ganzzahliger Programme. Wir diskutieren insbesondere jüngere und zum Teil unveröffentlichte Arbeiten. Ein Überblick mit vergleichbarem Fokus und Umfang war bislang in der Literatur nicht verfügbar.

Abgeleitet von einem Ansatz aus der Literatur entwickeln wir in Kapitel 3 ein gemischt-ganzzahliges Programm (MIP) zur Modellierung der praktischen Aufgabe. Das Modell ist von polynomialer Größe, die mittels LP-Relaxation gewonnene untere Schranke auf den optimalen ganzzahligen Zielfunktionswert jedoch beweisbar schlecht. Daneben formulieren wir ein binäres, sog. Set-Partitioning-Programm. Die Restriktionen erzwingen die Bedienung jedes Auftrags genau einmal; Variablen korrespondieren in diesem Modell mit Teilmengen von Aufträgen, die auf einer gegebenen Lokomotive zulässigerweise bedient werden können; im Allgemeinen exponentiell viele. Wir zeigen, dass die mittels LP-Relaxation gewonnene Schranke nicht schlechter als die des MIPs ist. Für beide Modelle stellen wir Erweiterungen in Bezug auf die Einbeziehung von Vorrangbedingungen zwischen Aufträgen vor.

Für die Lösung der LP-Relaxation der Set-Partitioning-Formulierung schlagen wir in Kapitel 4 die vorgenannte Spaltengenerierung vor. Kernstück des Kapitels ist das sog. Pricing-Problem, das jeweils zur Erzeugung einer neuen Spalte der Koeffizientenmatrix gelöst werden muss. Es beinhaltet die Konstruktion einer kostenminimalen Verkettung von Mustern, wobei für die Bedienung eines Auftrags zusatzliche reelle Kosten in Form der zugehörigen Dualvariablen des LPs anfallen. Zunächst beweisen wir, dass dieses Problem $\mathcal{NP}$-schwer ist. Zur exakten Lösung adaptieren wir Algorithmen zum Auffinden beschränkter kürzester Wege. Unser Markierungsalgorithmus zeichnet sich dadurch aus, dass – der Aufgabe entsprechend – nicht einzelne Knoten an bestehende Teillösungen, sondern ganze Muster, d.h. Sequenzen von Knoten angehängt werden. Nicht Erfolg versprechende Erweiterungen werden vermieden, indem über die aktuelle Dualinformation Schranken auf den bestmöglichen Wert solcher Erweiterungen mit der aktuell besten zulässigen Lösung verglichen werden. Das in diesem Kontext neue Verfahren stellt sich als sehr effektiv bei der Einschränkung des Suchraums heraus. Wie beweisen eine Approximationsgarantie für eine der als Ergänzung angebenen Heuristiken. Ganzzahlige Lösungen für die Set-Partitioning-Formulierung erhalten wir heuristisch, indem sich an die Lösung der LP-Relaxation ein Branch-and-Bound-Verfahren anschließt.

In Kapitel 5 diskutieren wir unsere Implementation. Mit zwei aus der Praxis und einem aus der Literatur stammenden Datensatz zeigen wir die Möglichkeiten und Grenzen unseres Verfahrens. Für praxisrelevante Daten lassen sich etwa 35 Aufträge für sechs Lokomotiven in wenigen Minuten optimal verplanen – dies ist die größte bisher optimal gelöste Instanz dieser Problemklasse. Damit eignet sich das Verfahren für kleinere Bahnen oder Teilbereiche größerer Bahnen. Alle Instanzen können in akzeptabler Rechenzeit ganzzahlig gelöst werden. Die Qualität der aus der LP-Relaxation gewonnenen unteren Schranke, und damit die unserer Lösungen, ist exzellent.

In einem letzten Kapitel 6 untersuchen wir theoretische Eigenschaften der eingeführten Muster. Unter vereinfachenden Annahmen geben wir zunächst Anzahlen verschiedener möglicher Verkettungen an; dies in Hinblick auf die Größe des Suchraums von Verfahren der impliziten Aufzählung zur Lösung unseres Problems. Wir zeigen, dass im Falle der in dieser Dissertation benutzten sehr einfachen Muster nur ein verschwindender Anteil der möglichen Lösungen betrachtet werden muss. Dennoch erhalten wir eine garantierte Qualität der Lösungen, wenn wir unser (einschränkendes) Konzept der Verkettung in einer allgemeinen Situation anwenden. Das Kapitel endet mit der Entwicklung einiger polynomial lösbarer Spezialfälle unseres Problems.

Wir schließen mit der Perspektive einer möglichen Implementation in der Praxis, einer Zusammenfassung und einem Ausblick auf mögliche sich anschließende Betrachtungen. Die umfangreiche Bibliographie mit 190 Referenzen, davon 114 jünger als 1990, verstehen wir als Beitrag von eigenständiger Berechtigung.

Curriculum Vitae

Persönliche Daten

Marco Lübbecke
geboren am 12. September 1971 in Helmstedt
verheiratet, zwei Kinder

Bildungsgang

1978 – 1982	Grundschule in Wolfsburg
1982 – 1984	Orientierungsstufe in Wolfsburg
1984 – 1991	Gymnasium in Wolfsburg, Abitur, Note 1,4
1992 – 1996	Studium der Mathematik mit Nebenfach Betriebswirtschaftslehre, Technische Universität Braunschweig, Diplom, Note „sehr gut“

Beschäftigungszeiten

1991 – 1992	Zivildienst
1994 – 1996	Studentische Hilfskraft bei Prof. Braß und Prof. Zimmermann
1996 – dato	Wissenschaftlicher Mitarbeiter bei Prof. Zimmermann